INDUSTRIAL CONTROL SYSTEM ENGINEERS

energy CONTROL

ENERGY CONTROL INTERNATIONAL PTY. LTD.
73 Eric Street Goodna Q 4300
Brisbane AUSTRALIA
Phone (07) 288 2455
Telex AA43778 ENECON

P O Box 12153, Wellington North,
NEW ZEALAND
Phone. 4-726462

THE EC-64 USERS' HANDBOOK

(Second Edition)

By T.H. Forster and G.D. Street

Edited by T.H. Forster

THE EC-64 USERS' HANDBOOK

(Second Edition)

By T.H. Forster and G.D. Street

Edited by T.H. Forster

Acknowledgements:

The Authors wish to thank Judy Forster for typing the manuscript, and Phil Keefe-Jackson for the photography.

T.H.F. and G.D.S.

T A B L E O F C O N T E N T S

CHAPTER 1

INTRODUCTION

CHAPTER 2

UNPACKING & SETTING UP

CHAPTER 3

OPERATION

CHAPTER 4

TUTORIAL / SYSTEMS MASTER DISK

CHAPTER 5

EC-BASIC

CHAPTER 6

EC-BASIC COMMANDS

CHAPTER 11

PERIPHERALS

CHAPTER 12

The EC MONITOR

APPENDICES

APPENDIX 1

ERROR MESSAGES

APPENDIX 2

ASCII CODES

APPENDIX 3

EC-64 TECHNICAL SPECIFICATIONS

CHAPTER 7

EC-DOS

CHAPTER 8

TEXT FILES and File Handling

CHAPTER 9

SUMMARY of DOS COMMANDS

CHAPTER 10

GRAPHICS

CHAPTER 1

I N T R O D U C T I O N

Objectives of the manual

Layout

How to use this Manual

Introduction to Computing

- realistic expectations
- what computers can and can't do
- what they do best, and worst

1: Introduction

OBJECTIVES:

The three main aims of this Manual are:

1. To enable the 'first time' computer user to make immediate use of the Energy Control Computer to run commercial computer programs (Software).

2. To provide an insight into the capabilities of the computer system by introducing the user to EC-BASIC and the EC-DOS Operating System.

3. To provide a comprehensive reference manual for all users of Energy Control EC-64 Computers.

To achieve these objectives, a simple tutorial style has been adopted. This is supplemented by technical reference material.

LAYOUT:

This manual consists of twelve chapters dealing with assembly and installation of the system as well as an introduction to EC-BASIC, EC-DOS and use of the 6502 Microprocessor. Comprehensive appendices have been included to supply essential technical information.

HOW TO USE THIS MANUAL:

Before turning your computer on for the first time it is advisable to read the first three chapters. A Tutorial / Systems Master Disk is supplied with this Manual and it should be used in conjunction with Chapter 4 to familiarize the user with the fundamentals of the system. After these chapters have been successfully negotiated, the natural progression will be determined by the interests of the user.

A chapter in which EC-BASIC is introduced at an elementary level is supplemented by a detailed summary of commands used in this language. This approach is designed to both encourage the novice and to satisfy the needs of the experienced computer programmer. A chapter on Graphics completes the comprehensive coverage of this powerful version of BASIC.

The Disk Operating System (EC-DOS) used with EC-BASIC and the 6502 Microprocessor is discussed in detail. This includes a broad coverage of file handling routines which will be useful to all programmers regardless of experience.

AN INTRODUCTION TO COMPUTING:

The expectations of the new computer owner are generally quite high. Computers will perform many tasks for you very efficiently and it is true to say that applications for computer usage are only limited by the imagination. It is not necessary to know how to write computer programs in order to use your computer to best advantage. In fact, it is advisable to use commercially available software until you become proficient in running the computer. In this way a knowledge of computer function and programming may be developed steadily while entertainment and efficient usage are gained from the computer.

Some areas where you may find ready use for your computer via readily available software are as follows:

The EC-64 Computer will facilitate your record keeping and financial planning. It will vastly improve your written work through word processing. It will entertain you and your family with the many and varied computer games available. Educational software packages abound for this machine and this takes you into the area of computer aided learning which is an exciting developing field of great potential benefit to you and your family.

All these things are possible but you should also have realistic expectations of your newly acquired computer. As you no doubt know, the computer cannot do anything for which it has not been programmed. If you are a newcomer to computing, programming skills will come gradually and not without some effort on your part. It is hoped that this Manual will provide a firm basis of computer knowledge and act as a ready-reference for use with the EC-64.

CHAPTER 2

U N P A C K I N G & S E T T I N G U P

Selection of operating site

Unpacking steps

The Big Warning

Description and Assembly of Components
- Computer Unit
  - CPU's
  - IC's
  - RAM
  - ROM
- Disk Drives
- Joystick
- Video Display Monitor

The Final Check

The EC-64 Computer:

FIGURE 1

Your new computer system is supplied to you in a number of boxes of various sizes. These boxes have one thing in common: the contents are liable to damage by rough handling, heat, dust and moisture. Treat the system with the care that such a sophisticated scientific instrument deserves and you will enjoy long and trouble-free use of your computer and it's accessories.

Selection of Operating Site:

The ideal site for your computer would be a quiet, convenient area away from passing traffic, fumes, humidity, dust and heat. It is not advisable to connect the computer to the same electrical circuit as a piece of high impedence apparatus such as a switching electric motor, a refrigerator or a power tool. A device of this type can cause an electric pulse which may trip the safety cutout in your computer and any programs in the memory at the time will be lost.

A desk or table with sturdy legs is ideal as long as it is stable and is large enough for the purpose. Take care to position the unit in an adequately lit area taking care to avoid reflections on the screen.

Unpacking and Checking the Components:

Carefully remove all of the components from their boxes. Count the parts and identify them by referring to the pictures below.

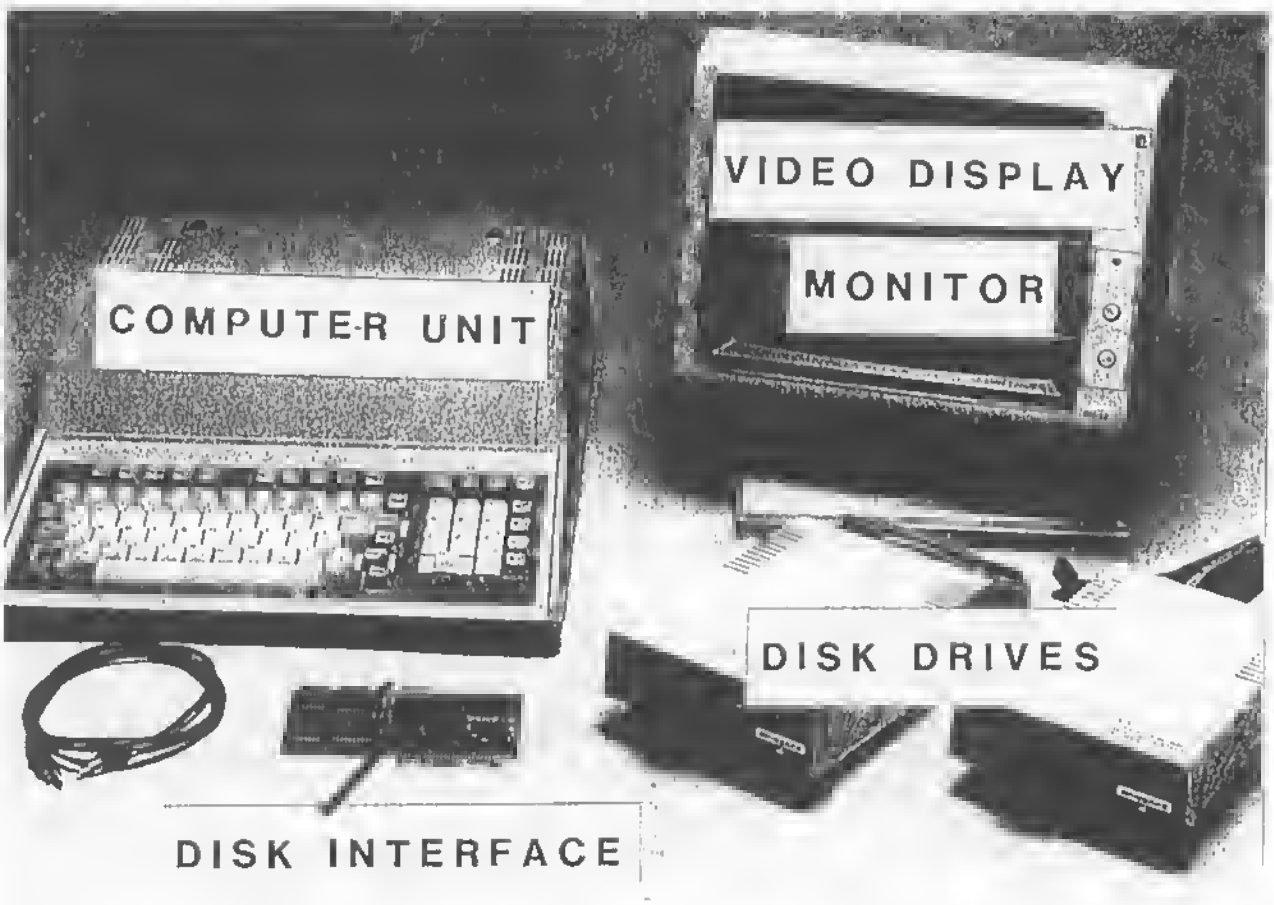


FIGURE 2

Our next step will be to discuss the various components, the functions they perform and how to go about connecting them.

Important Note:

THE COMPUTER MUST BE TURNED OFF AND THE POWER CORD DISCONNECTED BEFORE ANY COMPONENTS ARE INSTALLED, CONNECTED, DISCONNECTED OR UNPLUGGED.

Failure to do this may result in SEVERE DAMAGE both to the computer itself and to the interfaces and peripherals connected.

1. The Computer Unit.

The most obvious external feature of this unit is the keyboard. It has the standard QWERTY layout seen on most typewriters but it has as an additional feature a numeric keypad to speed up data entry. To the purist, the keyboard is not considered part of the computer proper. It is merely an input device. In some models the keyboard is supplied as a separate unit which has to be

attached to the computer unit by a flexible cable.

Place the Computer Unit in the desired position and open the upper part of the case by sliding the inspection panel backwards. Refer to the photograph below to identify the internal components of the computer. Pay particular attention to the power supply, the disk drives and their connection points and the 'slots' for interface connections.

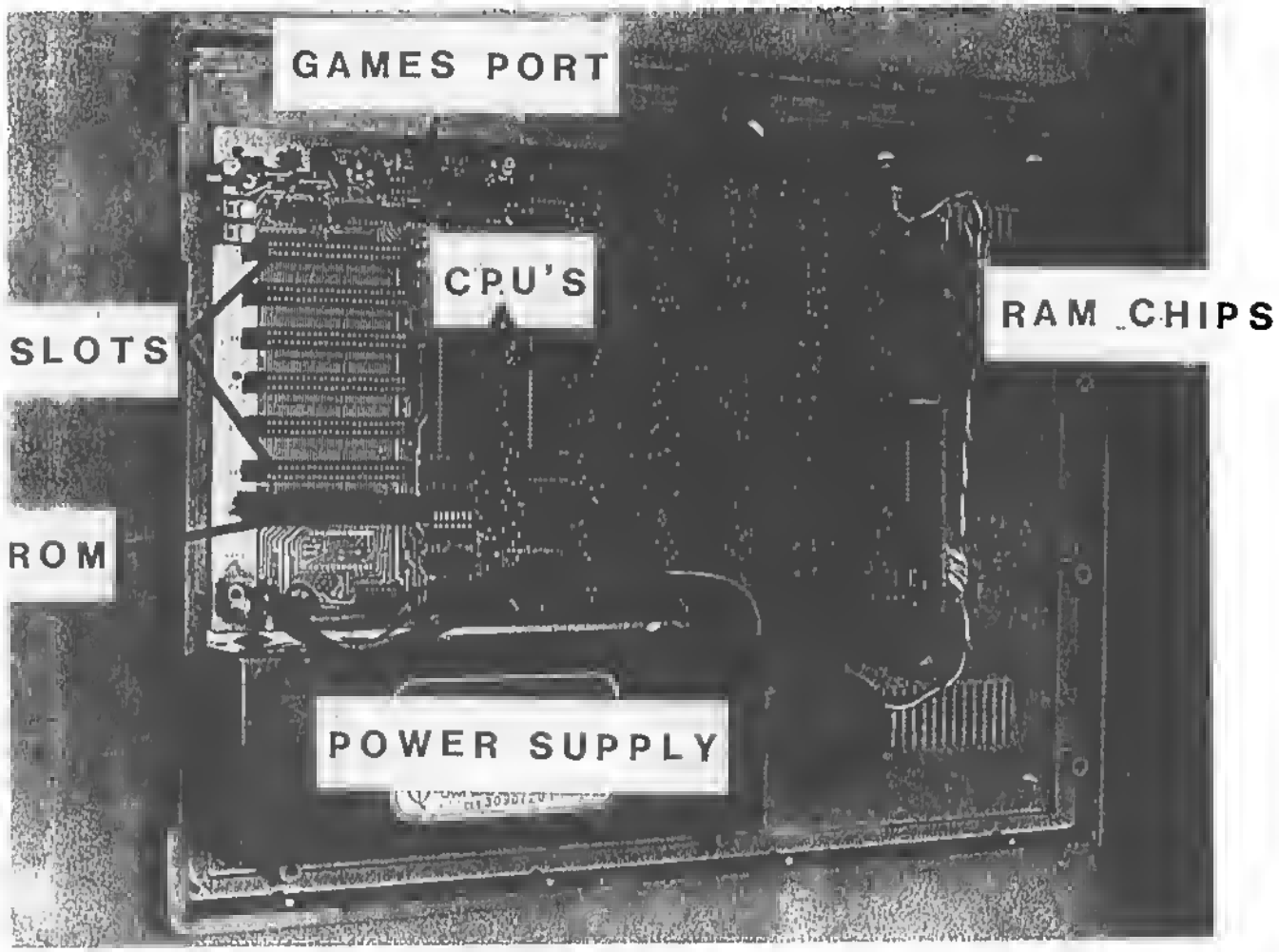


FIGURE 3

The first of the components to discuss are the microprocessors (often called Central Processing Units or CPU's). It is the microprocessor unit which is the real 'brains' of the computer. It is within this unit that all of the computations, calculations, decodeing and organisation of information is carried out. The EC-64 has two microprocessors, making it two computers in one. The first of these for you to identify is the one with the numerals '6502' amongst the identifying codes on its surface. It is the largest component in row 'H' on the main logic board or 'motherboard'. This CPU is the one which the system defaults to when the unit is switched on, and it is also the one which supports the EC-BASIC and FORTH languages.

The second microprocessor is identified by the letters 'Z80A' and is found in row 'F'. This CPU is the one which is called into operation when the CP/M operating system is loaded.

As well as the two relatively large CPU's, the motherboard contains numerous smaller integrated circuit boards (commonly called IC's or 'chips'). Each one contains numerous transistorized circuits. Collectively, they form the memory of the computer.

There are two types of memory within the computer. A large amount of memory has been set aside into which you may write or load your programs. These programs will be removed from the memory when the computer is turned off. This 'volatile' memory is called Random Access Memory or RAM for short. The EC-64 has 64K of RAM.

The second type of memory is permanent. It will not be lost when the computer is turned off, but under normal circumstances it cannot be changed by the user. It is this area of memory that contains the language EC-BASIC as well as the System Monitor program. Because our only access to this type of memory is when we read information from it, the name Read Only Memory (or ROM) is used. Physically, the ROMs for EC-BASIC and MONITOR are contained in the interface card plugged into Slot 0 of the motherboard. The EC-64 has 24K of ROM.

2. The Disk Drives

These small rectangular units (about the size of a hard cover novel) are recording devices which allow you to store your own programs onto magnetic disks. They also enable you to load programs from magnetic disks into the memory of the machine. The use and function of the disk drives is introduced in Chapter 3.

BE SURE THAT THE COMPUTER UNIT IS NOT PLUGGED IN BEFORE GOING ANY FURTHER.

In order to connect these units to the computer, it is necessary to make use of the interface card (the Disk Drive Controller card) supplied with the disk drives. After ensuring that the computer is turned off, insert the disk drive controller carefully in Slot 6 of the motherboard and press it down firmly until it is fully seated.

Remove the protective sheet of card from inside of the disk drives by releasing the locking lever.

Inspect the cable issuing from each drive. You will note that the end distant to the drive has a twenty pin box connector attached to it. These cable connectors are plugged into sockets marked DRIVE 1 and DRIVE 2 on the interface card. A small projection on one face of the box connectors ensure that the connections cannot be made the wrong way round. Identify the drive which is connected to the socket marked DRIVE 1. It is this drive which will always be activated first by the computer. If you have only one drive, it is necessary to have it connected to the DRIVE 1 socket always.

3. The Joystick (Games Controller).

This component is not essential to the functioning of the computer system, but as it is connected to the motherboard this would be a convenient time to attach it. The Joystick unit is slightly larger than a cigarette packet and about twice as thick. It has a ball-mounted leaver protruding from the top, two buttons on adjacent sides and a flexible lead with a plug attached. This unit is used mainly to control computer games but it has many other uses, depending on the program. To connect the Joystick plug the lead into the GAME I/O socket on the right hand side of the mother board.

At this stage, it is a good time to connect any other interface cards you have purchased. Install the interfaces with the IC chips on the right hand side, seating them carefully but firmly into the appropriate slots. For instance, the printer interface would go into Slot 1 and an eighty-column card would be put into Slot 3. See the instruction sheets supplied with each interface for details of installation, and refer to Chapter 11 for further information on peripheral devices.

When you are sure that all of the necessary internal connections are made correctly, replace the cover on the computer unit.

3. Visual Display Monitor.

This component resembles a small television set but it is capable of much higher resolution than a normal T.V. Place the Monitor on top of the unit. Plug one end of the connector cable into the socket on the back of the monitor and the other end into the socket provided on the back, right hand corner of the computer unit. (If you have not purchased a Monitor and you wish to connect the computer to a standard television you will require another piece of equipment called an RF Modulator. See your dealer about this).

A Final Check:

When you are sure all of the connections have been made correctly, examine the photographs below and compare your computer with the ones shown. If there are any differences recheck the above procedure.

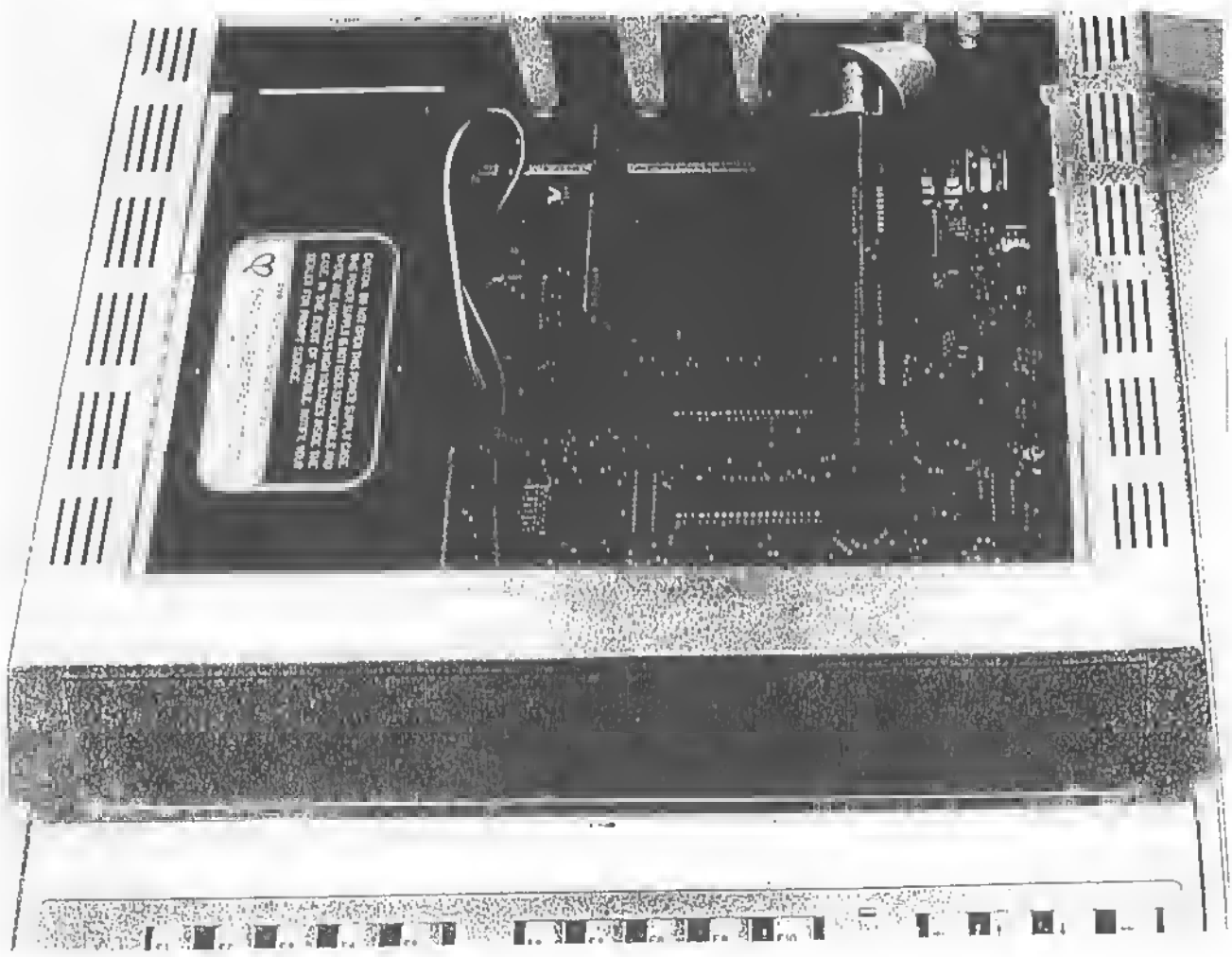


FIGURE 4

Plug one end of the power cord into the back of the computer and the other end into a wall socket. Connect the power cord from the Monitor to a convenient power point.

You should now be ready to 'power up' the computer for the first time.

CHAPTER 3

O P E R A T I O N

Powering Up
- Switches & indicators
- Description of a 'power up'

System Reset Menu - Explanation of each option
- Basic Cold
- Basic Warm
- Monitor
- Slots

Prompts and their significance

The Keyboard and Key Functions
- Discussion of each key function
  - Return
  - Shift & CAPS Lock
  - Control (Ctrl key combinations)
  - Editing, Cursor control
    - Arrows
    - Esc- I, J, K, M

The Disk Drive(s)
- Description & function
- Floppy Disks (diagram)
  - handling

Insertion of Disks
- write protection

Calling Up the Disk Drive

READ THE PREVIOUS CHAPTER BEFORE TURNING ON YOUR COMPUTER FOR THE FIRST TIME.

Powering Up

The sequence of events started when the computer is turned on is called 'powering-up'. Switch on the visual display monitor at the power point and then at the control switch on the front panel. The 'power-on' indicator light should come on. Allow the monitor to warm up for a couple of minutes before going any further.

Identify the power switch on the left hand side of the rear panel of the computer unit. Turn the computer on by first of all switching on the wall socket then the computer's power switch.

If all of the steps described in the previous chapter have been performed correctly a number of things should have happened when the computer was powered-up:

1. The red 'power on' indicator light should come on.

2. The above event is immediately followed by a short 'beep'.

[NOTE: If the power on light flashes on and off and no beep is heard it indicates a serious malfunction related to a component being incorrectly connected. If this situation occurs, turn the unit off and recheck all connections before attempting to power-up again. Be aware that some damage may have occurred if this has happened.]

3. The words SYSTEM RESET should appear at the top of the screen, with a number of other statements appearing below them (see below). This is referred to as the SYSTEM RESET MENU.

The screen should appear as follows:

```
SYSTEM RESET

(B)  BASIC COLD
(C)  BASIC WARM
(M)  MONITOR
(1-7) SLOT
```

This menu should appear whenever the unit is powered-up. It will also appear if both the CTRL (meaning 'CONTROL') and the RESET key are pressed at the same time. This will interrupt most functions of the computer. (CTRL-key combinations will be discussed later in the chapter).

Our next task is to discuss this menu and the options it offers.

The use of the word 'menu' implies that a selection may be made from this list, with some sort of palatable result for the user. In fact, it is from this menu that we select the mode of function of the computer.

Each of the options will now be discussed in turn.

The BASIC COLD Option:

If the 'B' key is pressed the computer will access the language called BASIC which is permanently stored in an area of its memory. The 'prompt' which indicates that EC-BASIC is in use (]) will appear, accompanied by a flashing cursor. It is now possible to write BASIC programs directly into the computer and operate them. However, it will not be possible to access the disk drives, either to store the program or to retrieve data. This is because the computer does not have a Disk Operating System (DOS) program permanently held in its memory. This program must be loaded from a disk using one of the other options from this menu which will be discussed later in this chapter. (See the next Chapter and also Chapter 7 for a discussion on DOS). The word 'COLD' in this option has meaning only when CTRL-RESET has been used to interrupt a program. Selecting this option in such a case causes the computer to 'forget' the program that was running and await instructions to be entered in the BASIC language. Remember that DOS would have been 'clobbered' by a CTRL-RESET if it had been loaded previously.

The BASIC WARM Option:

Pressing the 'C' key results in a similar situation to the one described for BASIC COLD except when CTRL-RESET has been used to interrupt a BASIC program. In this case the 'WARM' option allows you to drop back into the BASIC program that was in memory before the System Reset occurred. Note that the DOS program would still be lost meaning no disk storage would be possible.

The MONITOR Option:

Selecting this option by pressing 'M' results in directly accessing the EC-MONITOR mode and the identifying 'prompt' (#) appears on the screen. This mode allows you to examine Machine Code implementations of programs and memory areas. This is an area for advanced programmers and is discussed in Chapter 10.

The SLOT Option:

This is the most used option from this menu. It allows the user to access peripheral devices connected to one of the computer's communication slots. As there are effectively seven directly addressable slots we have a choice of up to seven devices which we may address. The disk drives are a peripheral device connected to Slot 6, so pressing '6' would result in activation of the disk drive attached to the Drive 1 socket. Using this option in this way allows us to load the DOS program from a disk in the disk drive. (Read the next Chapter for a discussion on this).

'Prompts' and Their Significance:

Two 'prompts' have already been mentioned, each identifying the language mode in operation in the computer. The use of these prompts acts as a reminder to the programmer in which language the computer expects the commands to be given. There are only three prompts which are commonly encountered when using an EC-64 Computer. They are:

] - means EC-BASIC language
> - means INTEGER BASIC language
- means MONITOR mode and MACHINE CODE

The Keyboard and Key Functions:

FIGURE 5

As mentioned in the previous chapter the keyboard has the standard QWERTY layout similar to most typewriters but you will notice that it has some extra keys. Because the computer can recognize a wide range of input signals we find that not only does each key send its own distinctive message, which is different in upper and lower case, but also combinations of keys can be used to input special codes. A standard code which most manufacturers adhere to has been established. This is called the American Standard Code for Information Interchange, commonly called the ASCII Code. A full listing of the character codes is given in Appendix B. In this section we will attempt to discuss some of the more important keys and key combinations.

Like many of the modern electric typewriters the EC-64 keyboard has an auto-repeat function on its keys. Pressing any key down and holding it will result in repetition of that character until the key is released.

The first important difference to note between a typewriter and a computer keyboard is that a zero is distinguished from the letter O by an oblique stroke. This is necessary because a computer recognizes the two as completely different entities because each has its own special code.

Another difference is a key marked RETURN. This key takes the place of the carriage return key or lever of a typewriter. Pressing RETURN signals to the

computer that an input is completed and processing of the contents of the input can commence. In some computers this key is lablled ENTER as it is the key used to pass information into the computer's memory.

The SHIFT and the CAPS LOCK keys have to be discussed together. The SHIFT key performs the usual transition between lower case and upper case characters when the CAPS LOCK is 'out', i.e. when the computer is in the lower case mode. However, if the computer is in the upper case mode the SHIFT key may be used to access some extra key functions. Unlike a typewriter, the characters which appear in the upper case position on the numeric keys will still only be accessed by pressing the SHIFT key even when the CAPS LOCK is 'in', i.e. in the upper case mode.

The CTRL (control) key has already been mentioned in conjunction with the RESET key. When the CTRL key is used it is pressed down and held while the other key to be used is pressed. The CTRL-RESET combination (which interrupts most computer functions, removes the DOS program from memory and returns the SYSTEM RESET menu to the screen) is just one of many CTRL-Key combinations which have specific purposes. One very useful one is CTRL-C, which can be used to interrupt many programs but leaves the contents of the memory intact. Some other combinations such as CTRL-D are used within computer programs to activate disk drive access commands. Some CTRL-Key combinations are used directly from the keyboard to activate some features of interface cards and peripheral devices. For details of these you must refer to the literature supplied with the appropriate card.

The FUNC (Function) key is another which has no counterpart on a typewriter. This key is used when writing computer programs to speed up the input of program lines. It is used in conjunction with the other keys in much the same way as the SHIFT key. For instance, holding down the FUNC key and pressing the 'N' key results in the word 'NEW' being printed on the screen. Each key has two accessible functions, the second of which is called by using a combination of the FUNC key and the SHIFT key. For instance, holding down both the FUNC and SHIFT keys together and then pressing the 'N' key will result in the word 'NEXT' being printed on the screen. A chart of functions is supplied with your computer and a summary of intelligent-keyboard operations is included at the end of this chapter.

The User-Defined function keys (numbered F1 to F10) are found in a row along the top of the keyboard. Each of these keys can store up to 48 characters in its own small RAM chip and retain the programmed characters for up to 5 years, due to a battery back-up. The keys are programmed by pressing CTRL-REPT, then the desired function key, i.e. F1, and then typing in the desired characters. Pressing the RETURN key twice ends the input.

The Numeric Keypad on the right of the keyboard is designed to speed up the entry of numeric data. Several mathematical function symbols are also supplied in this block to enhance its usefulness.

Editing and Cursor Control:

There are two sets of cursor control arrow keys provided on the keyboard. The first set to consider are those below the RETURN key, in the keyboard proper.

By the use of the right hand arrow, the computer has built into it the capacity to 'read' directly from the visual display monitor screen any character which the flashing cursor passes over in the forward direction (from left to right across the screen). Using the left hand arrow to 'backspace' has the opposite

effect, i.e. it causes the computer to 'forget' the characters passed over by the cursor. These features are very helpful when editing programs.

It is also possible to move the cursor around the screen without having it 'read' or delete any characters. This may be done in one of two ways:

Firstly, the four direction arrows situated above the numeric keypad will perform this function. This is the simplest way to move the cursor in the 'non-reading' mode.

Secondly, by using the ESC (Escape) key in conjunction with the keys I, J, K and M. Press the ESC key (but DO NOT hold it down) then press the I key to move the cursor up, the M key to move it down, or the J or K keys to move it left and right respectively. When the cursor is in the desired position, press the ESC key again to get back into the normal mode.

The Disk Drive(s) and Magnetic Disks:

A Disk Drive is basically a sophisticated recording device with many similarities to both a tape recorder and a record player. Like a tape recorder, it can be used to store information on a magnetic medium. Because the design allows the use of flat disks, access to any piece of information can be much more rapid than with a tape. (For details of Disk storage see Chapter 8, EC-DOS). Identify the locking lever, the insertion slot and the 'in use' indicator light.

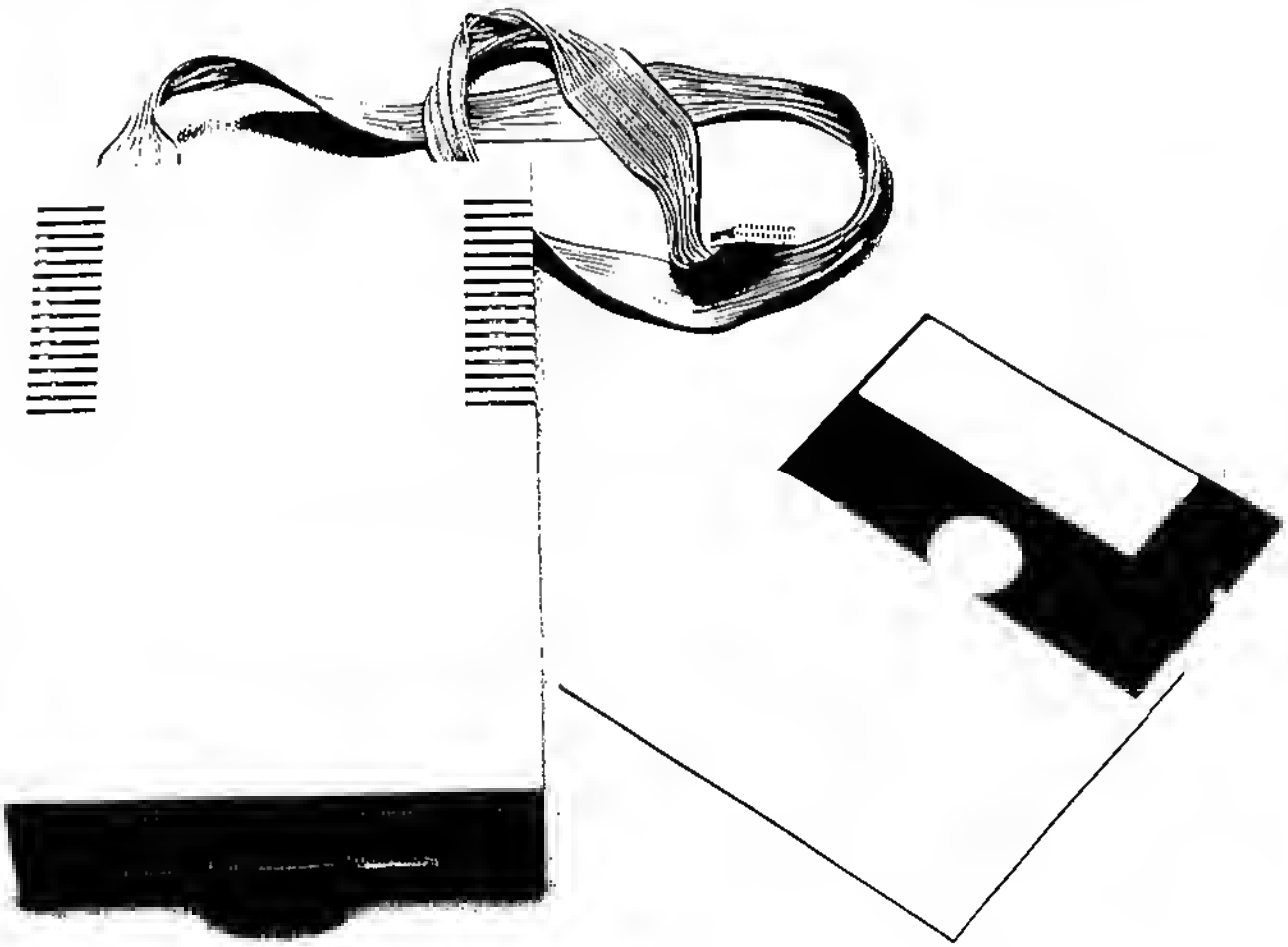


FIGURE 6

The magnetic medium used is a flat sheet of material of very similar nature to that used to make magnetic recording tape. Because the resultant disk is far from being rigid, the name Flexible Disk is often applied. Common usage has transmuted this into 'Floppy' Disk. These disks are encased in a black sheath of cardboard (which must never be removed or opened) and come supplied with a removable protective jacket.

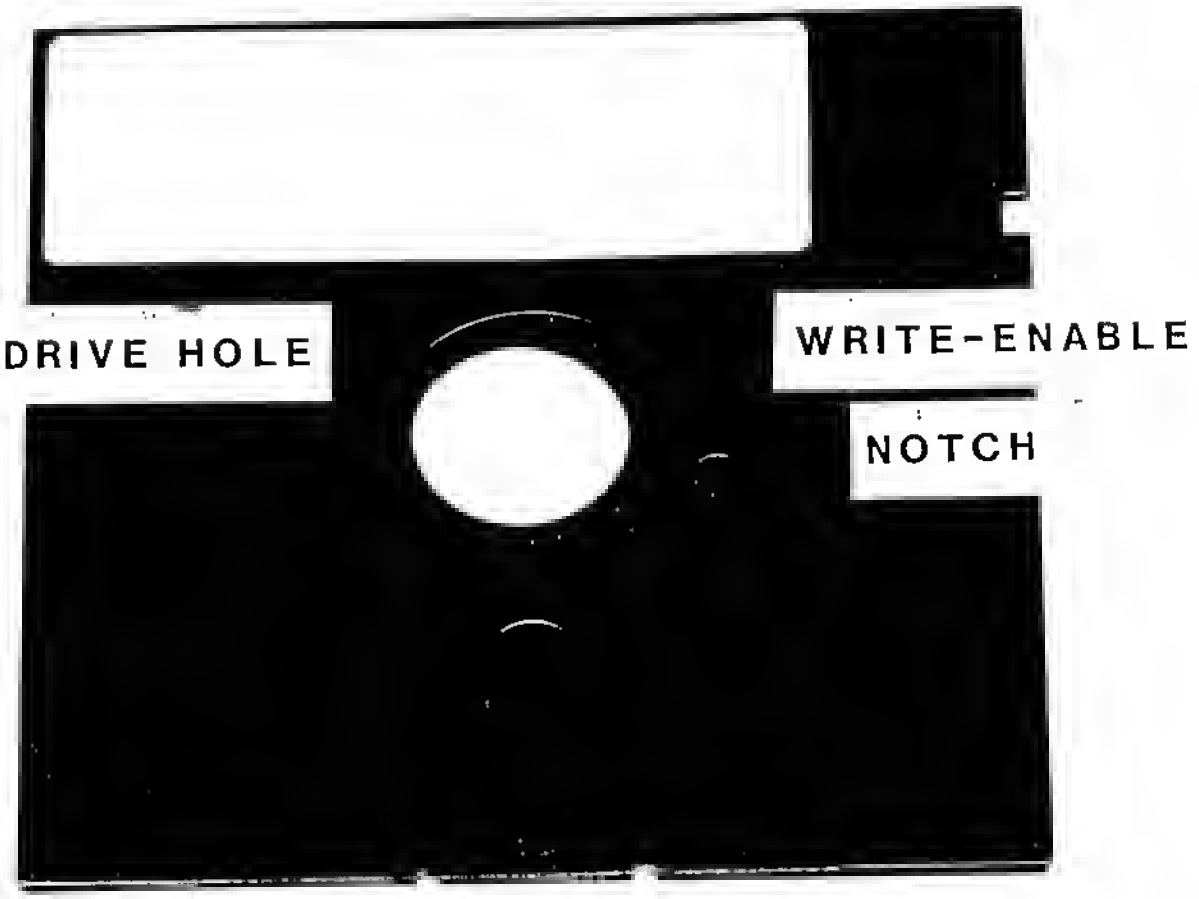


FIGURE 7

The features to note are the centre drive hole, the 'write enable' notch, the label, the oval shaped window which exposes the media, and the sector reference hole.

The 'write enable' notch needs some explanation. When this notch is present in the covering of a disk a mechanical sensor registers the fact and the mechanism of the disk drive is then capable of storing or writing information on that disk. If the notch is not present, or if it is covered up with a sticker (a 'write protection' sticker) then the disk drive unit is unable to write to that disk. In either case information is capable of being read from the disk. 'Write protection' of disks ensures that your valuable programs or files are not accidentally destroyed. Disks are 'write protected' by covering the notch completely with a gummed label.

Each of these disks is capable of storing large amounts of information. This makes it imperative that they be looked after correctly and be handled carefully to avoid the loss of valuable information or programs. There are ten

rules to be observed when dealing with 'floppy' disks:

1. Never touch the exposed surfaces of the media.
2. Do not bend or crush the disks.
3. Keep them away from magnetic fields - this means DON'T put them on top of the disk drives or the video display monitor.
4. Insert them carefully into the disk drives taking care not to bend or crease them.
5. Do not store them in hot or cold environments.
6. When not in use keep them in their protective jacket.
7. Never write on or rub out labels on the disk.
8. Never insert or remove disks from the disk drive while the 'in use' light is on.
9. Never turn off the computer with a disk in the disk drive.
10. Never CTRL-RESET while a disk is spinning in the disk drive.

Insertion of Disks:

Be sure that the red light on the disk drive is off. Note that disks must not be inserted or removed from the disk drive while the 'in use' light is on, as damage will occur to the disks. Hold your Systems Master Disk by the label and insert it into the disk drive, label side up, with the 'write enable' notch to the left hand side.

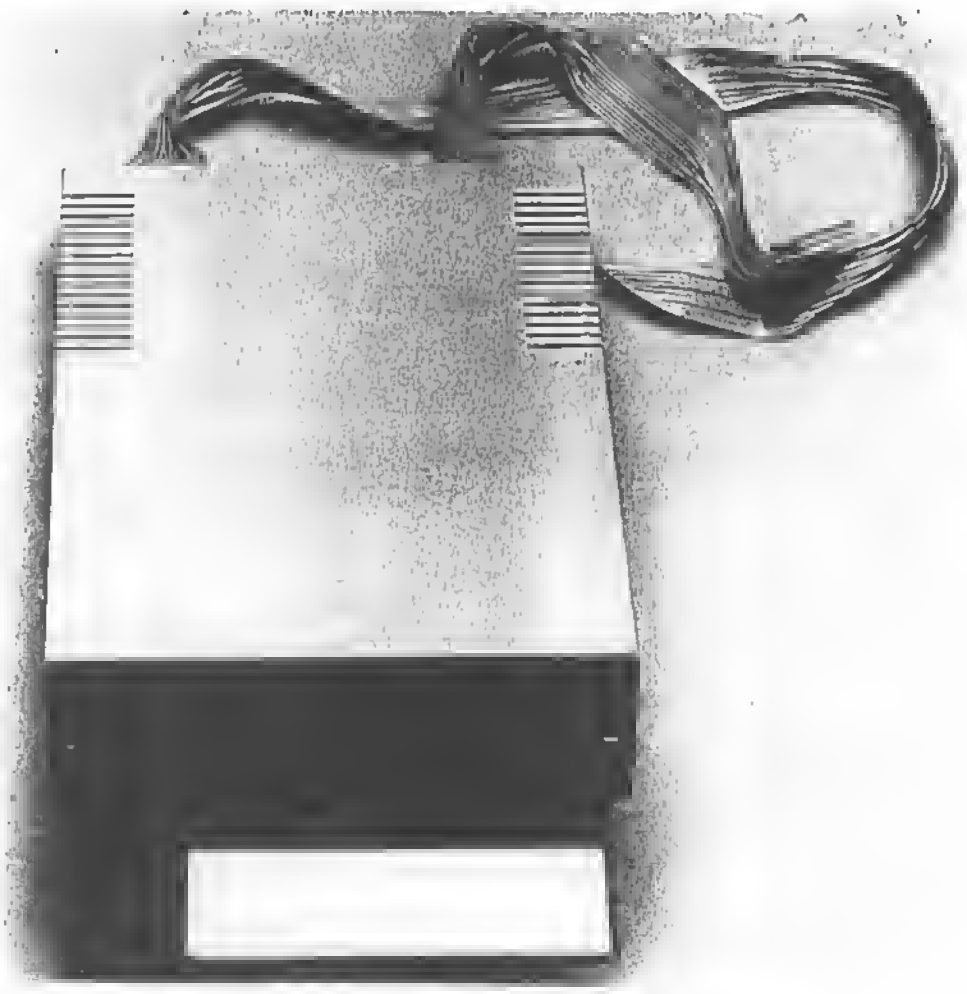


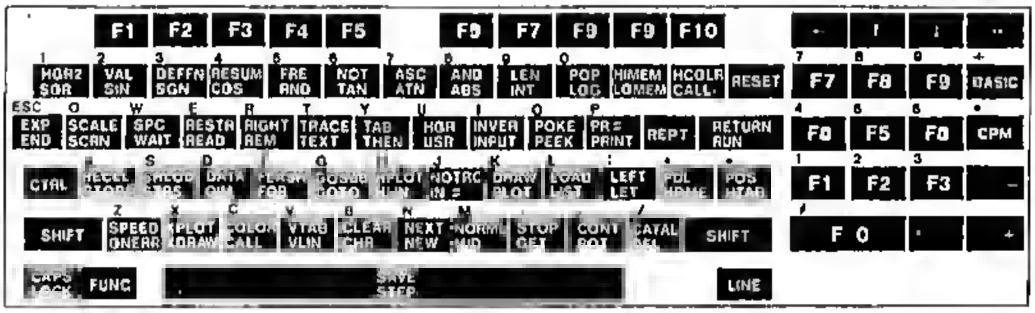
FIGURE 8

Be sure that the disk is fully inserted then press the lever down until it locks. This brings the recording heads into contact with the disk. The computer system is now ready to read from the disk. This will allow you to load a program into the memory of the computer and run it.

Calling Up the Disk Drive:

At this stage, the screen of the video display monitor should be displaying the SYSTEM RESET menu which was described earlier in this chapter. To activate the disk drive connected to the DRIVE 1 socket (which is the one you should have put your disk in) just press '6'. The 'in use' light should come on, a program will be loaded into the memory of the computer and will then run, resulting in a message being displayed on the screen.

If you have successfully negotiated all of this then you are ready to take the next step and run some programs from the Tutorial / Systems Master Disk. The next chapter deals with this.



Intelligent key board Operation

1. User Defined Function Key
   - F1, F2, F3, F4, F5, F6, F7, F8, F9, F10 is a User Defined Function Key.
   - Each key can storage 48 characters, and may be changed at any time. Keeps memory for 5 years.
   - At the same time, press "CTRL", "REPT" then release it. Press (any Key of F1, F2 . . . . . F10) Key. Input data. Press Return 2 times to terminate entry mode.
   - Press F1, F2 . . . F10 to output the data.
2. Using method for 94 BASIC, 94 CPM FUNCTION
   - At the same time, press "FUNC", "-" (numeric key) it will entrance into 94 CPM MBASIC FUNCTION
   - At the same time, press "FUNC", "+" (numeric key) it will entrance into 94 APPLESOFT BASIC FUNCTION
   - The Keyboard is in Applesoft Basic mode on Power up
3. Caps Lock
   - When you press the "CAPS LOCK" key, your computer will be in "abcde"
   - When you release the "CAPS LOCK" key, your computer will be in "ABCDE"
4. Auto Line Number
   - [illegible]
   - [illegible]
5. Function Key
   - [illegible]
   - [illegible]
6. Auto Repeat
   - [illegible]

CHAPTER 4

T U T O R I A L / S Y S T E M S M A S T E R D I S K

The First Step

An Introduction to EC-DOS
- Booting DOS
- CATALOGing a disk

Description of the screen catalog
- Symbols & Prefixes
- *, A, B, I, T, R

More about EC-DOS
- LOADing a program
- RUNning a program

A Small Step for Man
- LISTing
- SAVEing the program
- DELETEing a program

Copying Files and Duplicating Disks

The First Step:

The Tutorial / Systems Master disk supplied with this manual is a double sided disk which contains a series of programs designed to both help and amuse you.

Side A contains the 'housekeeping' programs, that is, those designed to help you with your tasks and to instruct you in the basics of programming. Side B has a 'slide show', games and demonstration programs to show you some of your computer's capabilities.

By using side A of the disk in conjunction with this chapter, some of the mysteries of computing will be revealed.

The first step is to 'power - up' the computer, so that the SYSTEM RESET menu appears on the screen. Next, put the Systems Master disk in Drive 1, with side A facing upwards (remembering to close the door of the disk drive). Press the key numbered '6'.

You will notice the red light on the front of the disk drive come on for a time, and then go out. A message will appear on the screen.
Congratulations! You have just 'booted' the EC-DOS.

An introduction to EC-DOS:
(For more details on EC-DOS, see Chapter 7)

'DOS' stands for Disk Operating System. It is simply a computer program that allows the computer to communicate with those interesting little devices called disk drives. The DOS is necessary to tell the drive where, and in what format to store the information on the magnetic disk. Perhaps more importantly it tells the disk drive mechanism how and where to find any particular program on the disk, and how to read it.

The DOS program must be loaded into memory before any communication between the computer and the disk drive is possible.

NOTE 1:
IF DOS HAS NOT BEEN BOOTED, YOU WILL NOT BE ABLE TO SAVE YOUR PROGRAMS TO DISK.

NOTE 2:
WHEN OPERATING IN EC-BASIC, IT IS ALSO POSSIBLE TO SAVE TO MAGNETIC TAPE. This method of storage is useful as a low cost back-up system for data or program storage. Also, if DOS has not been booted (or if it has 'crashed') the use of tape storage may save you a considerable amount of time.

The loading of EC-DOS into memory installs a very powerful utility into the computer. We will now use the Systems Master disk which you inserted into Drive 1 to demonstrate some of the features of DOS.

Type in the word CATALOG (note that all commands to the computer must be in upper case characters) and press the RETURN key (pressing this key enters the command or information into the computer).

If you have followed the instructions correctly, a number of things should have happened:

(i) The red light on the disk drive should have come on for a while, indicating that the drive is 'in use'. The light will then go out again when the drive stops spinning.

(ii) The words CATALOG and DISK VOLUME 254 should appear on the screen.

(iii) Three columns of letters and numbers should appear, the right hand one of which is the list of programs (files) on the disk.

CATALOG is a valid DOS command when used in the direct mode, after DOS has been loaded. If you have failed to 'boot' DOS (i.e. load the DOS program by booting a formatted disk) the computer's response to the word CATALOG is '?SYNTAX ERROR'. This is because the computer has not been told (programmed) to respond to that word.

Now, examine the display on the screen. You will notice that each program or file name is preceded by a letter (in some cases with an asterisk in front of it) and a three - digit number. Our next task is to consider what these mean.

The ASTERISK (*):
This signifies that the file is 'locked'. A locked file cannot be accidentally deleted or written over when disk storage is taking place.

The FILE TYPE:
The letter in the next column is an indicator of the file type. This is a necessary pointer to tell us in what form the file is stored and the nature of the computer language used to create that file. There are four common file types, identified by the letters A, I, B and T, as follows:

A - EC-Basic
I - Integer Basic
B - Binary
T - Text file

Occasionally a relocatable assembly language file, identified by the letter R, may be encountered.

The FILE SIZE:
A three - digit number is displayed with each file to indicate how long the file is, in terms of 'sectors'. The larger the file, the greater the number of sectors needed to store it.

The FILENAME:
This is the name by which a program is identified on the disk. It is by entering this name EXACTLY as it is displayed that we can retrieve the program from the disk.

More About EC-DOS:

The LOAD command:
This command is used to load a program written in EC-Basic from disk. Type in the words LOAD EXAMPLE and then press the RETURN key. The disk drive will become active for a few seconds, and then the prompt will reappear on the screen. The EC-Basic program called EXAMPLE is now installed in the memory of the computer.

The RUN command:
Type RUN and press the RETURN key. You will see that the program executes a series of commands, and then the prompt reappears. Now type RUN EXAMPLE and press RETURN. You will notice that the disk drive become active for a few seconds and then the program runs as before, ending with the return of the prompt.

You have learned an important lesson. If RUN is followed by a filename, the DOS will automatically LOAD and then RUN the program for you. If RUN alone is entered, the program in memory will be executed.

The BLOAD command:
Binary files (those prefaced by 'B' in the catalog) require this form of the load command. The command is given in the form BLOAD followed by the required filename.

The BRUN command:
This command, followed by the filename of a binary fine, will BLOAD and then BRUN that file.

Both of these commands perform similarly to the LOAD and RUN commands for EC-Basic.

A Small Step for Man:

LOAD or RUN the program called EXAMPLE again.

The LIST command:
When the prompt reappears on the screen, type the word LIST and press RETURN.

The list now displayed on the screen is the series of instructions, written in a computer language (in this case EC-Basic) which the computer executes when the file called EXAMPLE is run. This, then, is a 'computer program'.

By using the LIST command you are able to examine programs written in EC-Basic. This command is of great use to you when you are writing and 'debugging' your own programs.

The SAVE command:
Type SAVE EXAMPLE 2 and press RETURN. The 'in use' light on the disk drive will come on for a few seconds, indicating that the drive has performed some function. If you CATALOG the disk now, you will notice that a file called EXAMPLE 2 has been added to the previous list.

Here is a note of warning about the SAVE command: it must always be followed by a filename, but if that filename already exists on the disk, THE EXISTING FILE WILL BE WRITTEN OVER and destroyed.

The DELETE command:
Type DELETE EXAMPLE 2 and press RETURN. The disk drive will again become active, and then stop. If you now CATALOG the disk, you will see that the file called EXAMPLE 2 has been removed from the directory. DELETEing is possible for any file on a disk, as long as that file is not locked.

By this stage you should be capable of running any program you desire from the catalog on the Tutorial disk. It is a worthwhile exercise to run a few of the

entertainment programs on the reverse side of the disk. To do this, first turn the disk over, replace it in the disk drive and CATALOG it. Choose one of the programs from the directory and RUN it. Using the computer is really as simple as that.

Duplicating Disks and Copying Files:

PLEASE NOTE: Copying of software may infringe the Copyright laws. Check with your dealer if in any doubt.

As all magnetic media has a finite and limited life, making backup copies of your programs or files is a practice to be encouraged. To help you in doing this, a copying program is included in the utilities on Side 1 of the Tutorial / Systems Master Disk. It is suggested that you use this program now to make a backup copy of Side 1 of this disk.

Safety First! Write - protect the disk by placing a sticker or label over the notch in the casing of the disk (see Chapter 3).

After placing the disk in the drive with Side 1 uppermost, RUN the program called COPY. Answer the simple questions asked of you and then place a blank disk in the second disk drive (if you have two drives), or keep it ready for insertion (if you have only one drive). Note that the disk which is to receive the copy will be completely over - written, and any previous files on it will be destroyed. If you wish to proceed with the copy, press the RETURN key. All of the programs on the original disk will be copied.

In some instances, the copying of individual files on to a second disk may be required. In these cases, a file transfer program such as FID may be used. This type of program allows you to select which file or files you wish to transfer, and will add them to a disk which already contains files, without destroying any data on that disk. Such programs are marketed by many commercial software suppliers.

CHAPTER 5

E C - B A S I C

BASIC Programming

Immediate and Deferred modes
- Use as a calculator
- Deferred Mode
- Line Format
- Variables

Programming At Last
- Example Program 1

Debugging
- TRACE and NOTRACE

Explanation of Example Program 1

Example Program 2

Explanation of Example Program 2

Extending Example Program 2
- Subroutines

BASIC PROGRAMMING:

BASIC, an acronym for Beginners All-purpose Symbolic Instruction Code is a computer programming language which was developed by John Kemeny and Thomas Kurtz in 1964 at Dartmouth College, New Hampshire, U.S.A. as a simple language for newcomers to computing.

In this chapter we will demonstrate a few fundamental EC BASIC programming techniques to enable you to write your own programs on the EC64. This will be done by simple example programs illustrating some of the commands and functions of BASIC. For a fuller description of these commands and functions including syntax refer to Chapter 7 which lists all of them, grouped according to use.

Immediate Mode:

Before we delve into the arcane arts of programming, let's just take a moment to explain the differences between immediate and deferred modes as they relate to your EC64 computer. Immediate or, as it is sometimes called, direct mode allows you to use the computer as you would a calculator. O.K., take a deep breath and power up the beast. Refer to Chapter 4 on how to turn on your computer if you've suffered a memory lapse. Now call up EC BASIC by pressing the key marked B. If all's well you should recognize the ']' prompt which tells you you're in EC BASIC and the computer is awaiting your command. Now type in the following line exactly as it appears below.

```
PRINT 6+5
```

Press return and the computer responds by printing 11 on the screen. When no line number precedes a statement such as the example just given, and it is entered into the computer by pressing the return key, the statement is executed immediately. Hence this mode is termed immediate mode. Try a few more examples until you feel comfortable using the computer in this mode.

```
PRINT 8*3
PRINT 5+(4/2)
PRINT 16/2-3*2+1
PRINT (8/2-3)*((14-9)*2)
```

By the way, the arithmetic operators (+,-,*,/,^) employed by the EC64 may differ a little from those with which you are familiar so you should refer to Chapter 7 for a full explanation. Operator precedence (the order in which the computer processes these arithmetic operations) is also discussed there.

Multiple statements may be entered and executed in immediate mode.

```
FOR A=1 TO 10: PRINT A: NEXT
```

If you typed in the instructions correctly, when you pressed return the computer printed the numbers from 1 to 10 down the left side of the screen. This is an example of a simple 'loop' which will be explained more fully later in this chapter. If typing is not your forte you may use ? as a shorthand abbreviation of PRINT.e.g.

```
?5^2
?"5^2"=";5^2
?"THIS IS A STRING"
```

A 'string' as you have just seen, is a string of characters (including spaces) enclosed in quotation marks.

Deferred Mode:

The deferred or programmed mode is the usual mode in which the computer is used. In this mode the computer stores a set of instructions or program and obeys those instructions only after it is commanded to RUN the program.

A BASIC program consists of a sequence of statements. These statements provide the computer with step by step instructions which it follows to solve a problem or perform a particular task. The order in which statements are executed is determined by linenumbers. However, this sequence may be altered by the use of certain statements which cause the program to branch or jump to other sections of the program.

Line Format:

Just as the English language has certain rules of syntax and punctuation so has EC BASIC. The general format for a line in a BASIC program is as follows:-

LINE NUMBER	INSTRUCTION	EXPRESSION
650	PRINT	X*(INT(4/Y)-1)

In this example the line number 650 is followed by an instruction or verb PRINT which commands the computer to evaluate the arithmetic expression X*(INT(4/Y)-1) and print the result on the screen. The statement in this case is made up of the command PRINT and the arithmetic expression.

Multiple statements may be written for one line number if the statements are separated by colons e.g.

```
70 PRINT A: PRINT B: PRINT A+B
```

This causes the value of the variable A to be printed on the first screen line, the value of B to be printed on the second and the sum of A+B on the third line.

Variables:

In the example above, A and B are variables. You may think of variables either as the names we give numbers (arithmetic variables) and strings (string variables) or as 'pigeon holes' in which we store numbers or strings of text. 'A' is an arithmetic variable but 'A$' is a string variable. The dollar sign '$' identifies the variable as a string variable. Only the first two characters of a variable name are significant. Thus the computer would interpret FR$, FRANC$ and FRUIT$ as the same string variable. The first character must be an alphabetic character but the second character may be alphabetic or numeric e.g. AB, A1, C3, D, Z9$, F$ are all legal variable names. Be careful of including reserved words in variable names as this will make them illegal e.g. PASS$ is a legal variable name but PASSWORD$ is illegal as it contains the reserved EC BASIC word 'OR' and would be interpreted by BASIC as 'PASSW OR D$'.

PROGRAMMING AT LAST!

With that information under your belt it's time to write a simple program. Type 'NEW' and press return. NEW erases any previous program which may have been stored in the machine and thus prevents lines of the old program, whose line numbers you fail to use in the new program from being incorporated in your new program with subsequent disastrous results. Now type in the following program exactly as shown:

```
 10   REM EXAMPLE PROGRAM #1
 15   HOME: TEXT
 20   PRINT "DEMONSTRATION OF 'PRINT' COMMAND"
 30   PRINT: PRINT
 40   LET A=5: B=2: C=4
 50   S$="SUM": T$="TOTAL": LET T1$="THE"
 60   O$= "OF"
 70   PRINT T1$;" ";S$;" ";O$;" VARIABLES 'A' AND 'B' IS ";A+B
 80   PRINT
 90   PRINT
100   PRINT "A", "B", "C"
110   PRINT A, B, C
120   PRINT T1$, S$, T$
130   PRINT: PRINT
140   PRINT T1$+ S$+ T$
150   END
```

Now type RUN and press return. If all the statements were typed in exactly as shown then the screen should appear like this:-

```
DEMONSTRATION OF 'PRINT' COMMAND


THE SUM OF VARIABLES 'A' AND 'B' IS 7


A             B             C
5             2             4
THE           SUM           TOTAL


THESUMTOTAL
```

If your screen looks like this congratulations! It is indeed rare to enter a program without making a mistake. If you made a typographical error when entering the program, the output to the screen may be different or you may have received an error message.

Debugging:

A 'bug' is simply computer jargon for an error. Correcting these errors is termed 'debugging'. Error messages provide an aid to debugging. A comprehensive list of error messages and their meanings can be found in appendix H. If the computer returned an error message when you tried to run the program, look up the error message in appendix H. This will give you an explanation of the error that has occurred and may enable you to correct the bug immediately. LIST the line number containing the bug by typing 'LIST (linenumber)' and correct the mistake using the editing keys. Refer to Chapter 4 for details on editing program statements.

Some of the common typographical errors which occur when typing in programs include:-

1. Omission or substitution of punctuation marks.
2. Mistaking '0' for 'O' or 'I' for 'l' and vice versa.
3. Omitting spaces.
4. Spelling mistakes in commands.

A command which is very useful when debugging a program is TRACE. Type 'TRACE' and press return. You have now placed the computer in TRACE mode. Type 'RUN' and you will see each program line printed on the screen as it is executed. This is exceptionally helpful when tracing bugs in large programs. Type in 'NOTRACE' to turn off the TRACE mode. If you are still unable to find the error then LIST the program and compare each line with the original, paying particular attention to the common mistakes tabled above. Correct any mistakes you may find.

Now that you have the program running correctly let's examine the program in detail:

Line 10 is a REMark statement which is used for the purposes of explanation by the programmer. The computer ignores anything in the same linenumber after encountering REM although the characters following REM will still be printed when LISTed.

Line 15 clears the screen and sets the text mode.

Line 20 prints the string in quotes on the first screen line.

Line 30 prints two blank lines.

Lines 40, 50 and 60 assign various values or strings to numeric or string variables as appropriate. Note the optional use of the LET command.

Line 70 prints string variables and a string constant separated by spaces. Notice the use of the semicolon so that the characters are printed immediately following each other. Remember the computer recognizes a space as a character. Finally, the computer looks up the values assigned to variables A and B, adds them and prints the result.

Lines 80 and 90 print blank lines.

Line 100 prints the characters A, B and C. Notice that the use of the comma causes the computer to print these characters in consecutive tab fields.

Line 110 prints the values of the arithmetic variables A, B and C in consecutive tab positions.

Line 120 prints the strings represented by the string variables T1$, S$ and T$ in consecutive tab positions.

Line 130 again prints two blank lines on the screen. Note the separation of multiple statements by a colon.

Line 140 'concatenates' or adds the three string variables specified together to form one long string and prints this on the screen.

Line 150 causes the program to end.

Having mastered the simple techniques of assigning variables and formatting of screen output, let's move on to a simple demonstration of inputting data and controlling the program flow:

```
10 REM EXAMPLE PROGRAM #2
20 HOME : TEXT
30 HTAB 2: VTAB 3
```

```
 40 INPUT "HELLO. WHAT'S YOUR NAME ? ";N$
 50 HOME :T= 0
 60 PRINT :NU= INT(RND(1) *100) +1
 70 PRINT "I'VE CHOSEN A NUMBER BETWEEN 1 AND 100."
 80 PRINT
 90 PRINT TAB(4);"SEE IF YOU CAN GUESS WHAT IT IS ?"
100 T = T + 1
105 PRINT
110 INPUT "YOUR GUESS ? ";GN: PRINT
130 IF GN > NU THEN PRINT TAB(12);"THAT'S TOO HIGH!": GOTO 100
140 IF GN < NU THEN PRINT TAB(12);"THAT'S TOO LOW!": GOTO 100
150 IF GN = NU AND T > 10 THEN PRINT "THAT'S RIGHT! YOU GUESSED
    MY NUMBER": PRINT : PRINT "BUT IT TOOK YOU ";T; " GUESSES."
160 IF GN = NU AND T < = 10 THEN PRINT "FANTASTIC ";N$;" !":
    PRINT : INVERSE : PRINT "YOU MUST BE TELEPATHIC!": NORMAL:
    PRINT : PRINT "YOU GUESSED MY NUMBER IN ONLY ";T;" TURNS!"
170 PRINT : INPUT "OO YOU WANT TO PLAY ANOTHER GAME ? ";A$
180 IF LEFT$ (A$,1) = "Y" THEN GOTO 50
190 PRINT : PRINT "THANKS FOR THE GAME, ";N$
200 FOR I = 1 TO 6000 STEP 2: NEXT: HOME
210 END
```

Example program #2 is a simple guessing game in which the computer generates a random number and asks you to guess what it is. Let's examine the program line by line to see just how this amazing feat is performed.

Line 10 is a REMark statement which identifies the program for our own purposes. Remember the computer ignores everything on the same line after encountering REM so we can place comments here to clarify the program so that other programmers can follow what we have done.

Line 20 clears the screen and sets the text mode.

Line 30 positions the cursor two spaces in from the left margin on the third line from the top.

Line 40 prompts the player to INPUT his or her name which is stored in the string variable N$.

Line 50 clears the screen and sets a variable T to zero. We are going to use this variable as a counter. T will be used to store the number of turns the player has before guessing the computers number.

Line 60 prints a blank line and generates a random number NU with a value between one and one hundred.

Line 70 informs the player.

Line 80 prints a blank line.

Line 90 asks the player to guess the number. Note that the text is TABbed four spaces from the left to give a more pleasing appearance by centering the text on the screen.

Line 100 updates our counter T. You can see that the first time this line is executed T is assigned the value 1 which corresponds to the player's first guess. Subsequent guesses will increment our counter by 1 each time. In this way we can keep a record of the number of guesses the player has had.

Line 105 prints another blank line. Incidentally, these blank lines are inserted for cosmetic purposes and to make the text more readable on the screen.

Line 110 allows the player to INPUT his guessed number which is then stored in GN. Another blank line is also printed.

Line 130 compares the number the player has guessed, GN and the number the computer has generated, NU. IF GN is greater in value than NU then the computer informs the player that the number he guessed is too high. Finally, the program flow is interrupted by the GOTO statement which directs that line 100 will be executed next. This is known as a conditional branch. In this way, since the player has not correctly guessed the computers number, he is given another chance to guess the number. It should be noted that the GOTO statement is never executed unless the condition stipulated by the IF/THEN statement i.e. GN>NU is true. If that condition is false i.e. GN<NU, the next linenumber is executed and the instructions following THEN are ignored including multiple statements on the same line.

Line 140 compares the numbers once more. IF the condition GN<NU is true then the player is informed and program execution branches conditionally to line 100. Otherwise, the next line is executed immediately.

Line 150 compares the two numbers to see IF they are equal AND also checks if the player has had more than ten guesses. Notice that both conditions stipulated by IF must be true for the THEN part of the statement to be executed. IF both conditions are true the player is told he has correctly guessed the number and how many guesses he has had.

Line 160 compares the two numbers to see IF they are equal AND compares the number of guesses with 10 to see IF T is less than or equal to 10. IF both conditions are true THEN the player is congratulated via positive messages on the screen. The message 'YOU MUST BE TELEPATHIC!' is highlighted by displaying it in inverse video.

Line 170 asks the player if he wishes another game storing his INPUTed answer in the string variable A$.

Line 180 performs a conditional branch to line 50 to replay the game provided that the first character of the string variable A$ is Y. Thus the condition is true if the player's answer is YES, YEP, YUP, YAIR, YESSIR or simply Y.

Line 190 thanks the player for the game if his answer is anything other than something beginning with Y. That is, line 190 will be executed if the player answers N, NO or XENOLITH in response to the prompt in line 170.

Line 200 contains a delay loop in the form of a FOR/NEXT/STEP statement. This ensures that the thank you message remains on the screen until the computer counts to 6000 in steps of 2. The screen is then cleared by the HOME command.

Line 210 ENDs the program.

Well after all that you might like to type in the program and play the game for a while. Having come this far you deserve a break!

The simple example program #2 may be enhanced and made more interesting in a number of ways e.g. you could store the actual guesses that the player makes and print them out at the end of the game. This could be done by storing the guesses in what is known as an array. For example, we could substitute line 110 with:-

```
110 INPUT "YOUR GUESS ? ";GN(T): PRINT
```

GN(T) is an example of a one-dimensional numerical array. Arrays may have up to 88 dimensions. Each element of the array GN(exp) is a discreet variable and will store each guess made by the player. Notice that GN(1) corresponds to the first guess. The 2nd guess will be stored in GN(2) and so on. Arrays should be DIMensioned to reserve space in memory for the array, so add this line:

```
15 DIM GN(100)
```

The player should be able to guess a number between 1 and 100 in 100 turns! Actually 101 elements are DIMensioned for the array as the elements are numbered from zero to 100, giving a total of 101.

Substitute GN(T) for GN in lines 130, 140, 150 and 160 and add these lines:-

```
162 PRINT : INPUT "WOULD YOU LIKE TO SEE YOUR GUESSES ? ";RE$: PRINT
163 IF LEFT$ (RE$,1) = "N" THEN 170
164 FOR I= 1 TO T
165 PRINT "GUESS # ";I;" = ";GN(I)
166 NEXT I
168 PRINT
```

I'm sure you can think of other improvements so feel free to modify the program anyway you please. You'll learn a great deal more from implementing your own changes to the program. Modifications that you might consider could include giving familiar clues such as cold, cool, warm, hot in response to the player's guesses rather than high or low. For example, we could substitute line number 130 with :-

```
130 IF ABS(GN(T) - NU) > 30 THEN PRINT "YOU'RE COLD, ";N$: GOTO 100
```

This line evaluates the ABSolute difference between GN(T) and NU and if it is >30 prints the appropriate clue and executes a conditional branch. How would you implement the required changes?

One way we could do this would be to make use of the GOSUB command. This command directs program execution to a subroutine which when completed RETURNS to the statement immediately following the GOSUB. As an example, if we added the lines:

```
130 GOSUB 500
140 ON X GOSUB 600,650,700,750
145 IF GN(T) < > NU THEN GOTO 100
146 PRINT
500 REM SUBROUTINE TO DETERMINE PROXIMITY OF GUESS
510 X = INT (( ABS (GN(T) - NU) / 10) + 1)
520 IF X > 4 THEN X = 4
530 RETURN
600 REM SUBROUTINE
610 PRINT "YOU'RE HOT!"
620 RETURN
650 REM SUBROUTINE
```

```
660 PRINT "YOU'RE WARM."
670 RETURN
700 REM SUBROUTINE
710 PRINT "YOU'RE COOL."
720 RETURN
750 REM SUBROUTINE
760 PRINT "YOU'RE COLD!"
770 RETURN
```

Line 130 branches to the subroutine starting at line 500 where the proximity of the guess is calculated as follows:-

1. The ABSolute difference between GN(T) and NU is calculated:-
   ABS (GN(T) - NU)
2. This is divided by 10:-
   ABS (GN(T) - NU) / 10
3. 1 is added to the result:-
   (ABS (GN(T) - NU) / 10) + 1
4. The INTegral part of the result is determined:-
   INT ((ABS (GN(T) - NU) / 10) + 1)

Thus if the difference between GN(T) and NU is less than 10 then X will equal 1. If the difference is between 10 and 20 then X=2 and if the difference is between 20 and 30 then X=3. However if the difference is greater than 30, X=4 and line 530 RETURNs execution to line 140.

Line 140 causes the program to branch to the appropriate subroutine dependent on the value of X. If X=1 then execution passes to line 600. If X=2 then the program branches to line 650 and so on. The subroutines print the appropriate message before RETURNing to line 145 which has been included to branch to line 100 provided that the computers number has not been guessed.

Finally, another solution might be to try the following:- Add these lines

```
  11 DIM M$(4)
  12 FOR I = 1 TO 4: READ M$(I): NEXT
1000 DATA HOT,WARM,COOL,COLD
 530 PRINT "YOU'RE ";M$(X)
 540 RETURN
```

Delete lines 600 to 770 and 140. Much 'cleaner' and takes less effort to type in. In this case we have READ the DATA 'HOT' 'COLD' etc into a one dimensional textual array M$(I) and printed the appropriate message with just the one subroutine.

Two things before we leave you to it. By now the reason that we increment line numbers in steps of 10 will be obvious to you. It gives you room to add lines to the program later. This is very important as you will soon become aware! Also, it is a good idea to keep program lines short. It makes for easier editing.

Oh, by the way, these programs have been written to illustrate some BASIC techniques and do not necessarily represent good programming practice. For a more detailed BASIC programming tutorial, you should refer to such books as 'Illustrating BASIC' by Donald Alcock, published by Cambridge University Press or 'BASIC and the Personal Computer' by Dwyer, Thomas and Critchfield, published by Addison-Wesley Publishing Company.

CHAPTER 6

E C - B A S I C C O M M A N D S

Operator Precedence

Variables

System Control Commands

NEW	LIST
RUN	STOP
END	

Program Control Commands

FOR / NEXT	GOTO
IF / THEN	ON GOSUB
ON GOTO	ON ERR
RESTORE	REM

I/O Commands

INPUT	PRINT
HOME	GET
READ / DATA	SAVE

Math Functions

ABS	EXP
INT	LOG
RND	SGN
SQR	ATN
COS	SIN
TAN	

Memory Related and Special Purpose Commands

CLEAR	FRE
PEEK	POKE
CALL	HIMEM
LOMEM	DIM

String Commands

ASC	CHR$
LEN	STR$
VAL	MID$ / LEFT$ / RIGHT$

Graphics Commands

TEXT	GR
COLOR	HLIN
VLIN	PLOT
HGR	HGR2
DRAW	XDRAW
SHAPE	SCALE
ROT	

6: EC-BASIC Commands

This chapter has been composed to act as a brief reference for the EC BASIC language. In it, you will find descriptions of the syntax for all EC BASIC statements and functions along with simple examples which illustrate their use. Other examples may be found in earlier chapters and in the appendices.

The explanations accompanying the statements are by no means exhaustive. An effort has been made to keep these descriptions as simple as possible while ensuring that adequate information is given to enable the reader to employ the statements and functions in his own programs correctly. For an in-depth treatment of these commands and functions the reader should refer to one of the works on BASIC given in the bibliography.

A standard scheme for describing the format in which a command may be used has been adopted in this manual. The symbols and abbreviations used in the syntactic definitions are listed below.

/	A separator of alternatives.
[ ]	Brackets enclose optional items.
...	Denotes that the last item may be repeated
ln	Line number
avar	Arithmetic variable
svar	String variable
var	A variable - either arithmetic or string
aexp	Arithmetic expression
sexp	String expression
exp	An expression either arithmetic or string
aop	Arithmetic operator. i.e. + - * / ^
lop	Logical operator. AND/ OR/ NOT/ =/ >/ </ ></ <>/ >=/ <=
op	An operator either arithmetic or logical.

Operator Precedence:

The order in which operations are performed in evaluating expressions are predetermined by priority. Parenthesis have the highest priority and operations by them are executed first. Unary plus, minus, and NOT are executed next followed by exponentiation. Multiplication and division are next followed by addition and subtraction. Relational operations are then performed followed by the logical operators AND and OR which has the lowest precedence. Operators on the same level of precedence within the same expression are executed from right to left.

The order of operator precedence is as follows:-

()	Highest Precedence
+ - NOT	
^	
* /	
+ -	
> < >= <= < > =	
AND	
OR	Lowest Precedence

Variables:

Three classes of variables are catered for. These are real integer and string variables. Integer variables are denoted by attaching the % symbol eg. VAR%. Attaching the $ symbol to a variable signifies a string variable eg. VAR$. All other variables without tag identifiers (% & $) are real variables.

An integer variable must have a value which lies in the range -32767 to 32767. Real numbers must lie in the range -1E38 to 1E38 where 'E' stands for '*10A' i.e. $-1x10^{38}$ to $1x10^{38}$.

SYSTEM CONTROL COMMANDS

NEW FORMAT: NEW
EXAMPLES: NEW
10 IF PASS$ < > "ALPHA" THEN NEW
This command clears the current program and all variables stored in RAM. It may be used in either direct or programmed modes.

RUN FORMAT: RUN [ln]
EXAMPLES: RUN
10 RUN
120 RUN 100
If no line number is specified, RUN causes the execution of the program to start at the lowest line numbered statement. All variables and pointers are cleared prior to execution.

LIST FORMAT: LIST[ln/ln,/ln-/,ln/-ln/ln1,ln2/ln1-ln2]
EXAMPLES: LIST
LIST -50
LIST 200
LIST 110,
LIST 25-70
Writes or LISTS all or part of the program currently in memory on the screen. If no line number is specified, the entire program is displayed. If a line number is specified without a delimiter (- or ,), then only the specified line is listed. If a linenumber and either of the delimiters are used, the program is listed to the screen beginning with the specified linenumber. If two linenumbers separated by a delimiter are used, the program is listed starting at the first line number and ending at the second linenumber. If a delimiter followed by a linenumber is used then the program is displayed starting at the lowest numbered line and ending at the specified line. Listing may be halted and re-started by pressing CTRL-S or aborted by pressing CTRL-C. LIST may be used in either immediate or deferred modes.

STOP FORMAT: STOP
EXAMPLES: 80 STOP
The STOP command halts execution of a program and returns the computer to the immediate mode. In the above example, 8ASIC would stop execution of the program and print " BREAK IN LINE 80" on the screen.

CONT FORMAT: CONT
EXAMPLES: CONT
CONT causes the computer to continue execution of a program halted by a STOP END or CTRL-C at the next instruction after the one which halted it. CONT is normally used as a debugging aid in direct mode.

END FORMAT: END
EXAMPLE: 15000 END
END halts execution of a program and returns the computer to direct mode without displaying a message on the screen. It is usually the last statement in a program although its use is not obligatory.

TRACE FORMAT: TRACE
EXAMPLES: TRACE
10 TRACE
TRACE turns on a special mode which prints the linenumber of each statement on the screen prior to its execution. TRACE is used as a debugging aid.

NOTRACE FORMAT: NOTRACE
EXAMPLES: NOTRACE
65 NOTRACE
NOTRACE turns off the trace mechanism.

DEL FORMAT: DEL ln1, ln2
EXAMPLES: DEL 10,100
500 DEL 100,500
This command deletes the specified range of linenumbers from memory. Both linenumbers 1 and 2 are deleted. In deferred or programmed mode the range of linenumbers is deleted and the program is halted.

REM FORMAT: REM [text]
EXAMPLES: 400 REM SUBROUTINE TO CALCULATE INTEREST ON A LOAN
All characters following a REM statement including keywords on the same numbered line, are ignored by BASIC. This enables the programmer to insert explanatory comments within the body of the program as an aid to clarification. REM, by the way, is an abbreviation of remark.

PROGRAM CONTROL COMMANDS

FOR...TO...STEP FORMAT: FOR avar = aexp1 TO aexp2 [STEP aexp3]
EXAMPLES: FOR X = 1 TO 10
100 FOR I = B+18 TO B+8 STEP -1
The FOR/NEXT statement sets up a loop which is executed a stipulated number of times. The loop variable avar is set equal to a starting value aexp 1 and incremented by aexp 3 each time NEXT is encountered until the ending value of aexp 2 is reached. When avar = aexp 2 the program proceeds with the statement immediately following the NEXT statement. In this way, all statements occurring between FOR...TO...STEP and NEXT are repeated a predetermined number of times. If STEP is not used avar is incremented by 1 each time through the loop.

NEXT FORMAT: NEXT [avar]
EXAMPLES: NEXT I
100 NEXT X
50 NEXT
NEXT returns the program to the FOR...TO...STEP statement until the value of the loop counter is reached when execution passes to the statement following NEXT.

GOTO FORMAT: GOTO ln
EXAMPLES: 50 GOTO 1000
GOTO 200
GOTO transfers program control to linenumber ln unconditionally.

IF...THEN FORMAT: IF exp THEN [GOTO] ln
IF exp [THEN] GOTO ln
IF exp THEN statement
EXAMPLES: 10 IF A$ = "NO" THEN 400
60 IF Y/4 = INT (Y/4) THEN PRINT "LEAP YEAR"

```
30 IF X=8 THEN X=0 : GOTO 10
```

If the test condition (exp) is true then the program branches to the specified linenumber or the statement or statements following THEN on the same line are executed. If the exp is false execution passes to the next linenumber.

ON...GOTO FORMAT: ON aexp GOTO lnl [,ln....]
EXAMPLES: 110 ON X GOTO 300,400,500,600,750
Provided the value of aexp lies in the range 0-255 then a conditional branch is made in program execution. If aexp = 1 then the first line number in the list of linenumbers following GOTO is executed. Similarly, if aexp = 2 then control passes to the second linenumber in the list.

ON..GOSUB FORMAT: ON aexp GOSUB lnl [ln2...]
EXAMPLES: 430 ON Z GOSUB 1000,1100,1140,1300
ON GOSUB causes a conditional branch to one of a number of subroutines in the program. The subroutine executed is the one starting at the (aexp)th linenumber in the list after the keyword GOSUB.

ONERR GOTO FORMAT: ONERR GOTO ln
EXAMPLES: 10 ONERR GOTO 2000
This statement causes an unconditional branch to linenumber ln if a subsequent error occurs. This prevents program interruption when an error occurs.

GOSUB FORMAT: GOSUB ln
EXAMPLES: 50 GOSUB 900
GOSUB causes the subroutine starting at linenumber ln to be executed.

RETURN FORMAT: RETURN
EXAMPLES: 950 RETURN
When RETURN is executed program control returns from a subroutine to the next statement following the last encountered GOSUB statement.

RESTORE FORMAT: RESTORE
EXAMPLES: 510 RESTORE
This statement restores the pointer for the data list to the first item in the list. This prevents an OUT OF DATA ERROR occurring if an attempt is made to re-read the data list during execution of the program.

RESUME FORMAT: RESUME
EXAMPLES: 560 RESUME
RESUME transfers execution to the start of the linenumber in which an error occurred. It is usually the last line in an error handling routine.

POP FORMAT: POP
EXAMPLES: 550 IF X = 0 THEN POP
1010 POP
BASIC places the return address on the top of the stack when it executes a GOSUB statement. POP pops the top return address off the stack so that when a subsequent RETURN is encountered, program control passes to the statement immediately following the second last GOSUB executed.

LET FORMAT: [LET] var = exp
EXAMPLE: 10 LET SUM = A+B+C
40 AVERAGE = SUM/3
LET assigns a value to a variable. The use of LET is optional.

I/O COMMANDS

INPUT FORMAT: INPUT [STRING ;] var [,var...]
EXAMPLES: 10 INPUT "HOW MANY PLAYERS? ";NM
450 INPUT X1
700 INPUT "NUMBER OF DAYS? ";ND$
30 INPUT J%

INPUT halts the program and waits for data input from the keyboard. The user may be prompted for the type of data input required by use of an optional string. This command causes a question mark to be written to the screen if the optional string prompt is left out. Characters or numbers may be input as data into the appropriate variables.

HOME FORMAT: HOME
EXAMPLES: HOME
10 HOME

HOME clears the screen of text and places the cursor at the top left position on the screen.

FLASH FORMAT: FLASH
EXAMPLES: FLASH
20 FLASH

FLASH causes the text output on the monitor to oscillate between normal and inverse video.

INVERSE FORMAT: INVERSE
EXAMPLES: INVERSE
50 INVERSE

This command causes the screen text to be displayed in inverse video i.e. black characters on a white background.

NORMAL FORMAT: NORMAL
EXAMPLES: NORMAL
200 NORMAL

NORMAL sets the screen text to normal video mode i.e. white characters on a dark background.

SPEED FORMAT: SPEED = aexp
EXAMPLES: SPEED = 100
10 SPEED = INT (X/5)

This command sets the character output rate. The arithmetic expression must be between 0 (slowest) and 255 (fastest).

PRINT FORMAT: PRINT [[exp][,/;][exp]...]
EXAMPLES: 10 PRINT " TITLE"
560 PRINT "DEBT = $";DB
70 PRINT
100 PRINT "COL A", "COL B", "COL C"
90 PRINT X$;

PRINT used by itself causes a linefeed and return to occur. An arithmetic exp following PRINT causes the expression to be evaluated and the result printed on the screen. A string of characters enclosed in quotes following PRINT is printed exactly as it appears between the quotes. A variable (string or numerical) following PRINT causes the contents of that variable to be printed.

VTAB FORMAT: VTAB aexpr
EXAMPLE: 20 VTAB YY%
100 VTAB INT (Y/2)
40 VTAB 8
Provided the arithmetic expression equates to a value in the range 1-24, VTAB moves the cursor vertically to the line specified by the arithmetic expression.

HTAB FORMAT: HTAB aexpr
EXAMPLES: 15 HTAB X%
44 HTAB HO
10 HTAB 15
So long as the aexpr equals a value within the range 1-255 HTAB moves the cursor horizontally the number of spaces stipulated in the aexpr. 1 is the leftmost position of the current line while spaces 41-81 specify the line immediately below the current line and so on.

TAB FORMAT: TAB (aexpr)
EXAMPLES: 70 PRINT TAB (10); "TEXT TABBED RIGHT TEN SPACES"
200 PRINT TAB (X); "*"
TAB causes the print statement to begin writing to the screen in the column specified by the aexp which must lie in the range 0-255.

POS FORMAT: POS (exp)
EXAMPLES: 50 X=POS (X1)
90 JP=POS (A$)
400 PRINT POS (B%)
The current horizontal position of the cursor is given by POS command. Any legal expression may be used in the parentheses. POS unlike TAB and HTAB which count the leftmost or first horizontal position as one, numbers the first position as zero and returns a number in the range 0-255.

SPC FORMAT: (aexp)
EXAMPLES: 70 PRINT SPC(A-B); "HERE"
40 PRINT SPC(JJ%); "EXAMPLE"
60 PRINT SPC(167); "167 SPACES FROM LAST ITEM PRINTED"
SPC when used in a print statement, causes the item to be printed to appear on the screen indented from the left the number of spaces specified by the aexp. The aexp must evaluate to a number within the range 0-255.

GET FORMAT: GET var
EXAMPLES: 10 GET A$
70 GET X
GET causes the program to halt and wait for a one character input from the keyboard. In this case the character is not printed on the screen and there is no need to press the return key.

DATA FORMAT: DATA [string/constant][,string/constant][,..
EXAMPLES: 1000 DATA 10,23.65,9.713,4,0
400 DATA "AIR","GAS","OXYGEN","NITROGEN"
DATA creates a list of values to be assigned to variables by the READ statements. Elements in the data list are separated by commas.

READ FORMAT: READ var [,var...]
EXAMPLES: 50 READ A,B,C,D,E
10 READ FLUID$

Read assigns values from the data list to the appropriate variables in the same order as they occur in the data list. The first var after the READ command is assigned the first value in the data list. The second element is assigned to the second var and so on.

IN# FORMAT: IN# aexp
EXAMPLE: 40 IN#1
IN# establishes that input will come through slot number aexp. In this way peripheral devices may be selected to give input data. After IN# 0 input is expected via the keyboard. The aexp must be between 0 and 7.

PR# FORMAT: PR# aexp
EXAMPLE: 100 PR#5
PR# routes output via slot number aexp where aexp lies in the range 0 to 7. PR#0 transfers output to the screen.

MEMORY RELATED AND SPECIAL PURPOSE COMMANDS

CLEAR FORMAT: CLEAR
EXAMPLES: CLEAR
5 CLEAR
CLEAR sets all variables to zero and resets pointers and stacks.

FRE FORMAT: FRE (exp)
EXAMPLES: FRE (0)
1000 MEM = FRE (0)
10 PRINT FRE (0)
FRE gives the number of RAM bytes available (i.e. FREe) for your use.

PEEK FORMAT: PEEK (aexp)
EXAMPLES: 40 ER=PEEK (222)
PEEK gives the decimal value of the byte stored at memory address aexp.

POKE FORMAT: POKE aexp1,aexp2
EXAMPLE: 50 POKE 50,127
POKE causes the value of aexp2 to be stored in memory location aexp1. Values in the range 0-255 may be stored as one byte in one memory address.

HIMEM FORMAT: HIMEM:aexp
EXAMPLE: 10 HIMEM:50000
Enables the programmer to define the highest memory address available to a BASIC program. The aexp must be in the range -65535 to 65535.

CALL FORMAT: CALL aexp
EXAMPLE: 330 CALL -1994
This command causes BASIC to call or branch to a machine language subroutine beginning at memory location aexp.

LOMEM FORMAT: LOMEM:aexp
EXAMPLE: 15 LOMEM:
LOMEM allows the lowest memory address available to a BASIC program to be set to a value given by aexp.

USR FORMAT: USR (aexp)
EXAMPLE: 250 USR (4)
USR presents the value of the aexp to a machine language subroutine. The

result of the aexp is stored in an accumulator at address $9D to $0C. Control passes via a JSR to address $0A and provided a JMP instruction to the start of the machine language routine is found between addresses $0A and $0C the result of the routine is stored in the accumulator.

WAIT FORMAT: WAIT aexp1, aexp2[,aexp3]
EXAMPLE: 700 WAIT -16384,1,1
This statement causes a conditional pause to be implemented. A memory address is given by aexp1 and must be in the range -65535 to 65535 although this range is normally restricted to 0 to HIMEM. Aexp's 2 and 3 must be in the range 0-255. If only two aexp are specified then the contents of address aexp1 is ANDed with aexp2 and if a non zero result is returned then the wait is finished and execution resumes. If three aexps are used then the contents of aexp1 are XORed with aexp3 and the result ANDed with aexp2. The test is repeated until a non-zero result occurs and the wait is ended.

MATH FUNCTIONS

DEF FN FORMAT: DEF FN name (avar) = aexp1
FN name (aexp2)
EXAMPLES: 10 DEF FN RAD(X) = 3.1415926*X/180
200 Z = SIN (FN RAD (Y (1)))*SIN(FN RAD(J))
DEF FN enables user defined functions to be used in a program. Once the function has been defined using DEF FN that function may be used throughout the program as though it was an intrinsic function of the BASIC language.

ABS FORMAT: ABS (aexp)
EXAMPLE: 100 IF Z=ABS(Z) THEN PRINT Z;" IS POSITIVE"
ABS gives the absolute value of aexp without regard to the sign of the number.

ATN FORMAT: ATN (aexp)
EXAMPLE: 10 X=ATN (A)
ATN returns the arc tangent in radians of the aexp.

INT FORMAT: INT (aexp)
EXAMPLE: 30 WHOLE = INT (V/7.235)
INT returns the highest integer <= aexp

COS FORMAT: COS (aexp)
EXAMPLE: 30 R=COS (AG)
COS calculates the trigonometric cosine of aexp in radians.

SGN FORMAT: SGN (aexp)
EXAMPLE: 40 IF SGN (X) = -1 THEN PRINT X;" IS NEGATIVE"
SGN returns the sign of aexp. SGN equals +1 if aexp is positive, -1 if it is negative and 0 if it is zero.

SQR FORMAT: SQR (aexp)
EXAMPLE: 90 J = SQR(J)
50 HYPOTENUSE = SQR(S1A2 + S2A2)
SQR returns the square root of aexp provided aexp is positive.

SIN FORMAT: SIN (aexp)
EXAMPLE: 60 R = SIN (P1)
SIN returns the trigonometric sine of aexp in radians.

LOG FORMAT: LOG (aexp)
EXAMPLE: 300 C = LOG(NU)
LOG gives the natural logarithm (LOG_e) of the aexp.

TAN FORMAT: TAN (aexp)
EXAMPLE: 210 A = C*TAN (AG)
TAN returns the trigonometric tangent of the aexp in radians.

EXP FORMAT: EXP (aexp)
EXAMPLE: 30 H=EXP (Z)
EXP returns the exponential or natural antilogarithm of the aexp i.e. (2.718289) to the aexpth power.

RND FORMAT: RND (aexp)
EXAMPLES: 90 DICE = INT(RND(1)*6)+1
50 ZZ = RND(-1)
RND returns a random number in the range 0 to 1. The number may be equal to zero but is always less than one. If the aexp is negative then following RND functions with positive arguments will produce a repeatable sequence of random numbers i.e. pseudo random numbers. If aexp equals zero then the random number generated will be the same as the random number generated immediately prior to it. Positive aexp's greater than zero produce new random numbers each time.

STRING FUNCTIONS

ASC FORMAT: ASC (sexp)
EXAMPLE: 40 X = ASC ("BULL")
70 Y = ASC (AB$)
ASC gives the ASCII code number for the first character in the string expression.

CHR$ FORMAT: CHR$ (aexp)
EXAMPLE: 10 PRINT CHR$(69); CHR$(69-2); CHR$(90/2); CHR$(54); CHR$(52)
CHR$ is the opposite of the ASC function in that it returns the character whose ASCII code is the integer value of the aexp which must lie between 0 and 255.

LEN FORMAT: LEN (sexp)
EXAMPLE: 40 NUM = LEN (A$)
LEN returns the length of a string. The total number of characters including spaces, numbers and punctuation are counted.

STR$ FORMAT: STR$ (aexp)
EXAMPLES: 60 PC$ = STR$ (4001)
200 X = STR$ (V1)
The STR$ function turns the arithmetic expression into a string.

VAL FORMAT: VAL (sexp)
EXAMPLE: 70 PRINT VAL (TOTAL$)
400 PRINT VAL ("10 DOWNING ST.")
VAL gives the value of a numeric string. This function is the opposite of the STR$ function.

MID$ FORMAT: MID$ (sexp, aexpl[,aexp2])
EXAMPLE: 10 A$ = "SUPERMAN": B$ = MID$(A$,3,4): PRINT B$
MID$ returns the substring starting with the aexpl character of aexp2 characters from the middle of the string sexp. If the optional aexp2 is not used all characters from aexpl to the end of the string are returned.

LEFT$ FORMAT: LEFT$ (sexp,aexp)
EXAMPLE: 20 A$ = "SUPERMAN": B$ = LEFT$(A$,5): PRINT B$
This function returns aexp characters from the left side of the string sexp i.e. the first aexp characters of sexp.

RIGHT$ FORMAT: RIGHT$ (sexp,aexp)
EXAMPLE: 30 A$ = "SUPERMAN": B$ = RIGHT$ (A$,3): PRINT B$
This function returns aexp characters from the right side of the string sexp i.e. the last aexp characters of sexp.

+ (CONCATENATION)
FORMAT: sexp + sexp
EXAMPLE: 10 A$ = "BAT": B$ = "MAN": C$ = A$+B$: PRINT C$
+ is used to concatenate or join two or more string expressions.

GRAPHICS COMMANDS

TEXT FORMAT: TEXT
EXAMPLE: 500 TEXT
TEXT invokes the normal text screen format of 24 lines of 40 characters placing the prompt and cursor on the twenty fourth line

GR FORMAT: GR
EXAMPLE: 150 GR
Low resolution graphics mode is established by this command. Resolution is set to 40 X 40 pixels and a four line text window is enabled at the bottom of the screen.

COLOR FORMAT: COLOR = aexp
EXAMPLE: 160 COLOR = 3 : REM PURPLE
The COLOR command determines which colour is used when plotting in low resolution graphics and aexp may have any of sixteen values from 0 to 15

PLOT FORMAT: PLOT aexpl, aexp2
EXAMPLE: 400 PLOT X,Y
PLOT causes a pixel of x-coordinate aexpl and y-coordinate aexp2 to be illuminated on the low resolution screen provided aexpl lies in the range 0 to 39 and aexp2 lies between 0 and 47. If the COLOR of the pixel has not been stipulated then it defaults to COLOR = 0. X and Y-coordinates are measured from the upper left pixel which has the coordinates (0,0).

HLIN FORMAT: HLIN aexpl, aexp2 AT aexp3
EXAMPLE: 650 HLIN X1,X2 AT Y
HLIN causes a horizontal line to be drawn on the low resolution graphics screen from pixel aexpl, aexp3 to pixel aexp2, aexp3. As in the PLOT command x-coordinates must lie in the range 0 to 39 while y-coordinates must lie in the range 0 to 47 and the colour is determined by the most recently executed COLOR statement.

VLIN FORMAT: VLIN aexp1, aexp2 AT aexp3
EXAMPLE: 700 VLIN Y1, Y2 AT X
VLIN causes a vertical line to be drawn on the low resolution graphics screen from position aexp3, aexp1 to position aexp3, aexp2.

SCRN FORMAT: SCRN (aexp1, aexp2)
EXAMPLE: 880 HUE = SCRN (X,Y)
In low resolution graphics mode, SCRN gives the COLOR code of the pixel aexp1, aexp2 where aexp1 may have any values from 0 to 39 and aexp2 may have a value of from 0 to 47.

HGR FORMAT: HGR
EXAMPLE: 1000 HGR
HGR enables a high resolution graphics mode giving a screen with 280 horizontal pixels and 160 vertical pixels and a text window of four lines at the bottom. Page one of memory from 8K to 16K is displayed.

HGR2 FORMAT: HGR2
EXAMPLE: 250 HGR2
HGR2 establishes a high resolution graphics mode (280x192 pixels) without a text window and displays page 2 of memory (16K to 24K).

HCOLOR FORMAT: HCOLOR = aexp
EXAMPLE: 950 HCOLOR = 1 : REM GREEN
HCOLOR determines the colour of the pixel to be illuminated on the high resolution graphics screen. The arithmetic expression may have any of eight values from 0 to 7.

HPLOT FORMAT: HPLOT aexp1, aexp2
HPLOT TO aexp3, aexp4
HPLOT aexp1, aexp2 TO aexp3, aexp4 [[TO aexp,aexp]...]
EXAMPLES: 500 HPLOT X1,Y1
750 HPLOT TO X2,Y2
660 HPLOT X1,Y1 TO X2,Y2 TO X3,Y3 TO X4,Y4 TO X1,Y1
HPLOT causes a pixel of x-coordinate aexp1 and y-coordinate aexp2 to be illuminated on the high resolution screen. HPLOT TO causes a line to be drawn from the last pixel plotted to pixel aexp3,aexp4.

PDL FORMAT: PDL (aexp)
EXAMPLE: 440 IF PDL(0) < 128 THEN REV = REV/2
PDL gives the value (0 to 255) of the game control paddle aexp. The arithmetic expression must lie in the range 0 to 3 otherwise unexpected results may be obtained.

DRAW FORMAT: DRAW aexp1[AT aexp2,aexp3]
EXAMPLE: 600 COLOR=1: ROT=16: SCALE=100: DRAW AT X,Y
610 DRAW M
If only one aexp is used DRAW causes the aexpth shape in the shape table to be placed on the high resolution screen at the position stipulated by the last DRAW, XDRAW or HPLOT encountered. Arithmetic expressions 2 and 3 define the x and y coordinates respectively of the shape to be drawn.

XDRAW FORMAT: XDRAW aexp1[AT aexp2,aexp3]
EXAMPLE: 700 XDRAW S1
400 XDRAW M1 AT X,Y
XDRAW is the same as DRAW except that the colour in which the shape is drawn is the complementary colour to that already existing at the pixels drawn to.

ROT FORMAT: ROT = aexp
EXAMPLE: 490 ROT = 0 800 ROT = INCLINE
ROT specifies the orientation of the shape to be displayed on the high resolution screen. By use of the ROT command a shape may be rotated through 360 degrees. The aexp determines the degree of rotation.

SCALE FORMAT: SCALE = aexp
EXAMPLE: 490 SCALE = LARGE
370 SCALE = 56
SCALE is used to define the size of a shape via the aexp which must lie in the range 0 to 255.

CHAPTER 7

E C - D O S

DOS and Disk Storage
- Tracks
- Sectors

Loading EC-DOS

The HELLO Program

The INIT Command

More about the HELLO program

Deferred Mode DOS Commands

More DOS Commands

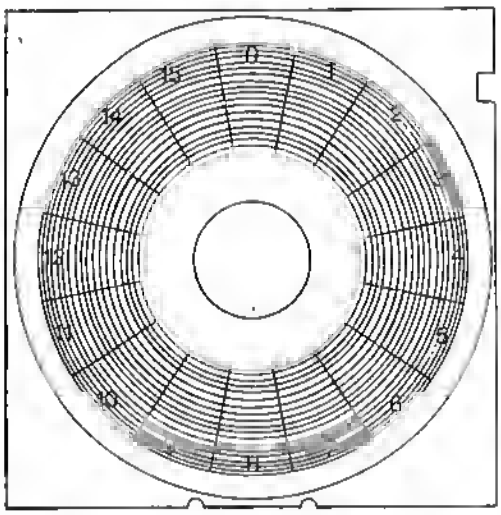
7: EC-DOS

The Disk Operating System (DOS) has been introduced in chapter 4. The aim of this chapter is to explain the function of DOS and to acquaint the user with the commands and features which make DOS such an indispensable part of the Computer Progammer's tool kit. (NOTE: For detailed formats of all of the DOS commands, see Chapter 9, EC-DOS Commands).

DOS and Disk Storage:

When information is stored on disk it is stored as a series of analog impulses held in a magnetic medium, in much the same way as a tape recording. In fact the flexible or 'floppy' disks which are used are of similar material to a magnetic tape. The advantage of disk storage lies in the speed of access to any particular piece of information, anywhere on that disk.

In order to locate and recognise specific data, the DOS program 'formats' the disks in a characteristic way. It divides the disk into sixteen 'sectors' radially, and into thirty - five 'tracks' concentrically (see diagram).



The tracks can be likened to the grooves in a record. The analogy is imperfect, however, because the tracks do not spiral in to-wards the centre as grooves in a record do. They form a series of thirty- five concentric circles, each of which is identified by its distance from the centre of the disk.

The sectors are like the spokes of a wheel, radiating out from the centre. The DOS program numbers them in order to be able to identify them later. The numbering of the sectors is done independently of any index holes or marks in the disk meaning that the DOS program can locate for itself any sector independent of the disk 'hardware'. Disks formatted in this way are said to be 'soft sectored'.

It is obvious then that each track is divided into sixteen separate sectors which are like cells into which we may put information. Each track sector can contain up to 256 'bytes' of information but if less than this amount is stored in any one track sector, then the rest of that sector cannot be used for storage of another file, i.e. it is reserved for that file.

A simple calculation will show you that there are 560 sectors on a disk and therefore we should be able to store 560 x 256 'bytes' of information, i.e.

about 143 K (or Kilobytes). However, the DOS program itself takes up some space, as does the catalog and some other related information. This leaves 496 free sectors for you to use, giving about 126 K of disk storage. As the memory of your computer in its standard form is 64 K, you can see that by using just one disk for storage of information you can expand the capacity of your computer by nearly two hundred percent.

Loading DOS:

The first step in the use of DOS is always to load the DOS program into the computer. This is done by 'booting' any disk which contains the DOS program, in this case the Tutorial / Systems Master disk supplied with this manual. The procedure for this has been described in chapter 5. Briefly, you must get the System Reset menu on the screen and then, with Side 1 of the Systems Master uppermost in the disk drive, press '6'. When the 'in use' light on the disk drive goes out and the prompt appears on the screen DOS has been loaded into its special area of the memory. Here it will stay, untouched by any programs that you load or write or by any command you give, until either the computer is turned off, or the CTRL (control) key and the RESET key are pressed together. The CTRL / RESET combination causes the SYSTEM RESET menu to appear on the screen and DOS to be eliminated from memory. In this case, you must load DOS all over again.

Now, remove the System Master disk from the disk drive and replace it with a blank disk. Please note that the next set of procedures described will destroy any files on the disk which is in the disk drive. Be sure that you do not use a disk which has files on it that you wish to keep.

To gain maximum benefit from this chapter it is suggested that you work through the series of exercises described. Just leave the disk in the disk drive and read on.

The HELLO Program:

Before we can impress the DOS program onto the blank disk in the disk drive, we have to prepare a file for DOS to call when the disk is 'booted'. A simple program in basic is all that is needed. It may consist of only one line, i.e.

```
10 PRINT "HELLO"
```

or it may be a large and complicated program.

Some variations on the HELLO program will be discussed later in this chapter. For the present, it is suggested that you follow the procedure and format given below.

(i) Type NEW and press RETURN. This command clears all programs (except DOS) from the memory of the computer (see chapter 6).

(ii) Enter the following program into the computer, remembering to press RETURN at the end of each line:

```
5 REM: HELLO
10 HOME
20 VTAB 14: HTAB 16: PRINT "HELLO"
30 PRINT: PRINT "SLAVE CREATED BY (Your Name) ON (Date)"
40 PRINT: PRINT "64K EC-64 SYSTEM"
50 END
```

This program has some important features for you to note. It identifies the disk as a 'slave' disk (as distinct from a 'Master' disk). It also identifies the owner and will tell you the age of the disk.(This may be important to you, as the disk media have a finite life.) Also, the habit of identifying the type of computer system used is to be encouraged, as this will help to identify the version of BASIC used. RUN the program to be sure that it executes correctly.

The INIT Command - INITializing a disk:

There are two main points to note when INITializing a disk using EC-DOS. They are:

1. The EC-DOS Systems Master Disk must have been 'booted' prior to this procedure being performed.

2. The INITialization process is a TWO STEP procedure. For those familiar with some similar types of DOS programs, this is a point to remember.

The First Step:

When you are satisfied that the 'Hello' program is 'debugged', type INIT HELLO and press RETURN. The 'in use' light on the disk drive should come on and a series of whirring noises should be heard for about a minute. What you have just done is to make the computer perform the complex task of formatting a disk.

Briefly what happened was:

(i) The disk was spun at a constant speed, and divided into 'tracks' and 'sectors'.

(ii) The sectors of the outer three tracks (tracks 0, 1 & 2) were all reserved for the DOS program, which at this stage has not been installed. In the process, the filename you used in conjunction with the INIT command has been recorded in a special location and will be RUN whenever the disk is booted.

(iii) The hingepin of the whole disk, the Volume / Table of Contents (VTOC) was written in track 17, sector 0. It is this one sector and its VTOC message that allows us to find the specific sectors where files are held. It is the 'reference mark' of the disk.

(iv) The remainder of track 17 is reserved for the CATALOG of the disk i.e. the filenames.

[It is interesting to note that the catalog is stored on the 17th or middle track of the formatted disk. This is the first track read when the disk is booted. By positioning the catalog here it means that the heads in the disk drive will have to travel less distance, on the average, to find any one program.]

(v) The BASIC program that we called HELLO was written into sectors of track 18, taking as much room as was needed.

The Second Step:

Type CALL 4096 and press RETURN.

The disk drive will again become active for a few seconds. The DOS program has now been loaded into the reserved sectors of tracks 0, 1 and 2. You have now successfully INITialized a disk, and it is ready for you to use.

There are still a few things to say about the INIT command. For instance, it is the only command that allows you to assign a VOLUME NUMBER to a disk. Any number between 1 and 254 may be used in the format:

INIT HELLO,Vn (where n = 1 to 254)

If no volume number is stated, DOS supplies the default value of 254.

More About HELLO:

The name 'HELLO' need not be used to identify the HELLO program to the DOS. Whatever name is used in conjunction with the INIT command will be remembered by the DOS program on that disk and will be called on to RUN when the disk is booted.

Instead of DOS calling a simple identification program such as the one used in our example, it is possible to INITialize the disk with any program you wish to RUN when the disk is booted. For instance, you may write a computer game program and decide to dedicate the whole disk to it. If you INITialize the disk with that program, the game will run automatically when the disk is booted. Such programs are said to be 'SELF BOOTING'.

Commands to RUN or LOAD other programs on the disk, or even to CATALOG the disk, may be included in the Hello program. The next section will deal with the special syntax needed to do this.

Deferred Mode DOS commands:

Commands that access the disk drives may be included in virtually any program. This use of DOS allows us to effectively expand the memory of the computer by either storing data on the disks as it is generated, or by loading programs from the disk as they are required.

The method of giving DOS commands within a program is quite simple. As with all computer commands, however, the syntax is all important. The format must be as follows:

```
50 PRINT CHR$(4);"DOS Command"
```

The CHR$(4) statement is CONTROL-D. The PRINT command preceding a CONTROL-D tells the computer to treat the statement in parenthesis following the delimiter (;) as a DOS command. (For an explanation of the PRINT and CHR$ commands, refer to chapter 6.)

To further demonstrate how to use DOS in this way, it is suggested that you make a small modification to that HELLO program you INITialized the disk with previously. We will modify the program to automatically CATALOG the disk when it is run.

The HELLO program should still be in the memory of the computer. (If it isn't,

boot the disk and it will be put there). LIST the program. Then type:

```
50 PRINT CHR$(4);"CATALOG"
60 END
```

If you LIST the program again now, you will note that the old line number 50 has been replaced by the line which includes the DOS commands.

Type SAVE HELLO and press RETURN. The modified program will have been saved to disk and will have replaced the original one. When the program is RUN (or the disk is booted) the program will call up the disk catalog of files for you.

By using this simple syntax in a program, we can easily RUN or BRUN or LOAD or BLOAD other programs from disk. (For details of the format of these commands see Chapter 9, EC-DOS Commands).

More DOS Commands:

In Chapter 4 the DOS commands LOAD, RUN, SAVE and DELETE were the means by which you were introduced to EC-DOS. These commands alone own will take care of most of your needs as far as disk access is concerned. The aim of this section is to introduce some other commands which will help you with your 'housekeeping'.

The RENAME Command:
This command allows you to change the name of any file on a disk. To demonstrate a couple of points about this function, we suggest you again use the disk that you INITialized a little while ago. Be sure that DOS has been booted (by pressing '6' from the SYSTEM RESET menu, with the disk in the drive, remember) and when the prompt re-appears type RENAME HELLO, BONJOUR and press RETURN. If you then CATALOG the disk, you will find that the name HELLO has disappeared and BONJOUR has taken its place. The program will LOAD and RUN under its new name, but will not answer to its old name. You have now created yourself a problem, as the DOS program loaded onto that disk was told to look for (and RUN) a program called HELLO. If the disk was booted in its present state, DOS would display a FILE NOT FOUND message on the screen. The point of this demonstration is that the RENAME command is not a respecter of personnages. It will RENAME anything to virtually anything, even if the change causes future problems for both you and DOS. Note that if the filename you choose to RENAME to already exists on the disk, the function will be carried out any way, leaving you with two different files with the same name. Very embarrassing! The only alternative is to RENAME one of them again. Oh yes - don't forget to RENAME BONJOUR, HELLO before you try to boot that disk again.

The LOCK command:
LOCKing a file renders it inaccessible to any of the DOS commands that could alter or eliminate that file. Type LOCK HELLO and press RETURN. If you CATALOG the disk you will notice that the filename now is preceded by an asterisk, denoting a LOCKed file, e.g.

```
*A 002 HELLO
```

The file will still LOAD and RUN successfully but it will not DELETE or RENAME. If you try these commands, the FILE LOCKED message is sent by the DOS and is displayed on the screen. The same result is seen if you try to SAVE a program which has a LOCKed filename already on disk. Clearly, this is a useful command. LOCKing prevents the unwitting destruction of existing disk files. Any type of disk file may be LOCKed.

The UNLOCK command:
This is the logical opposite of the LOCK command. Typing UNLOCK HELLO does just that, and renders that file accessible again to all DOS commands.

The VERIFY command:
This command is used to check that a file is stored correctly on the disk and ensures that the file is capable of being read by the computer. It is good practice to VERIFY all files after SAVEing or copying. To try this command, type VERIFY HELLO and press return. The disk drive will become active for a few seconds, then stop. If the file is satisfactory, the prompt will reappear on the screen . If there is a bad sector in the disk, or the file for some reason has not copied or saved correctly, then an I/O ERROR message is seen.

The PR# Command:
This command tells the computer to send information to a particular 'slot', numbered from 0 to 7 (see Chapter 3) where an output device is interfaced. To try this out, be sure that your INITialised disk is in the drive then type PR#6 and press return. This will cause the disk drive to be activated and the booting of the disk in the drive.

If you have a printer interfaced into slot 1, you will find that PR#1 will send any output from the computer to the printer. This includes any screen display, unless it is graphics. You will find that a special technique is needed to put graphic images onto the printer (see the printer's own instruction book, or the interface card's instructions for details).

PR#0 returns the output to the video screen only. The video screen is attached to slot 0 in the 40-column display mode.

The IN# Command:
Input from peripheral devices interfaced into slots 0 to 7 is called up by this command. As a disk drive may be considered an input device, IN#6 will boot the drive just as PR#6 did. Unlike PR#1, the IN#1 command will not access the printer. It will cause the computer to look for an input from the device in slot 1. When it doesn't find one, it will return control to the keyboard and the prompt reappears on the screen.

These two commands are obviously useful for the input and output of data. However, they do not allow us to output data to disk for storage, or to select a particular piece of information from disk for loading into the computer's memory.

To make best use of the disk drives we need to be able to store and retrieve more than just 'computer programs'. The acquisition, storage, retrieval and output of data, both in raw and processed form, represents an essential element of efficient computer usage. To do this we must take our next giant leap, and create Text Files.

The next Chapter introduces this aspect of EC-DOS.

CHAPTER 8

T E X T F I L E S a n d F I L E H A N D L I N G

An Introduction to Text Files

MAXFILES

MON and NOMON

Types of Text Files

Sequential Text Files
- Commands, Syntax & Defaults
  - CtrlD
  - OPEN
  - DELETE
  - WRITE
  - READ
  - CLOSE
  - APPEND
  - POSITION

Random Access Files
- Commands, Syntax & Defaults
  - OPEN and the L parameter
  - WRITE, READ and the R parameter
  - The B Parameter

EXEC Files
- Creation and use

8: Text Files

An Introduction to Text Files:

A simple definition of a text file is a file that has no executable statements and consists entirely of text and data. An example would be the names, addresses and telephone numbers of your friends or business acquaintances.

Some simple routines in BASIC which will allow you to create these types of files will be discussed later in this chapter. Before we can do this a few more commands need to be introduced.

The MAXFILES Command:

When DOS is booted areas of memory are reserved to act as 'buffers' for incoming and outgoing information during disk access. It is necessary to buffer information, i.e. store it temporarally, because of the speed differential between the computer memory which is quite fast and the disk reading or writing mechanism which is relatively slow.

DOS sets a limit on these reserved areas determining both their size and number. For each buffer, 595 bytes of memory are reserved. The default value for the number of buffer areas is 3. This means that up to three files may be in use at any one time. By using the MAXFILES Command we can set the number of buffers to be anything from 1 to 16 and thus control the total number of files which may be in use simultaneously. For example by entering MAXFILES 6 the number of active files would be set at 6 and about 3.5 K of memory would be tied up just to handle the input and output of those files. Remember that when DOS is booted the next time the default value of MAXFILES 3 will be executed.

The MON and NOMON Commands:

MONitoring communication of data between the memory and the disk drive can be very useful in 'debugging' file handling programs. You even have a choice of what you can monitor. Commands (such as OPEN and CLOSE) which operate the access to a file may be monitored. Input from the disk or Output to the disk can both be monitored as well. All 3 may be monitored together, singly, or in any combination by using MON followed by the letters C, I, or O or all 3 separated by commas as delimiters e.g.

```
MON C, I, O  (monitors all commands, inputs and outputs)
MON I, O     (monitors inputs and outputs only)
MON C        (monitors control commands only)
```

NOMON is the logical opposite of MON. It turns off the monitoring of the Commands, Input and Outputs as required and uses similar syntax to the MON command.

Examples of the use of these monitoring functions will be given later in the chapter.

Types of Text Files:

There are two main types of text files to consider, Sequential Text Files and Random Access Text Files. These two filetypes are essentially very different from one another in structure.

A Sequential File may be likened to notes made in a note book. Each note made reserves just enough room for itself but can be of any length. Any subsequent

entries may start immediately after the last entry. While we can store large amounts of information in this way, finding any particular piece of information may be quite difficult unless we are very familiar with the notebook.

A Random Access File in contrast may be likened to cards in a filing system where each card is kept for the records of one particular subject. The cards would be of fixed dimensions, meaning that there would be a limit to the amount of data able to be stored on each card and each card would occupy the same space whether it was full or empty. This takes more space but it allows us to locate, expand or update any particular piece of information much more easily.

This discussion on files is not intended to be all-encompassing. There are several excellent works on this topic to which the interested reader may refer. The aim is to introduce the commands necessary for their use and, more importantly, to stimulate the interest and imagination of the reader.

Sequential Text Files:

Special commands are necessary to create, store and retrieve all text files. One cannot simply write in a text file in the same way that a BASIC program is entered, directly from the keyboard. A small but carefully structured input routine written in BASIC is necessary. This means that the commands which operate files are not DIRECT commands, operating only in the deferred mode, i.e. only from within a program. Some examples of these programs are provided on Side 1 of the Systems Master disk. They will be referred to throughout this section to illustrate various points and applications.

The OPEN Command:

This command is used to signal the beginning of a file. It must be accompanied by a filename, be preceded by a CTRL-D (control-D) and be executed only from within a program. Remember that these constraints apply for all of the following commands. For detailed formats see the 'EC-DOS Commands' section at the end of this Chapter.

The WRITE Command:

After a file has been opened, a number of options are available to us. One of these is to WRITE a file to the disk. It is this command which takes the place of the SAVE command when files are to be stored.

The DELETE Command:

When a file is OPENed in preparation for WRITEing it is a good practice to issue a DELETE command before the WRITE command is given. This removes any other files with the same name as the new file. This is necessary because WRITEing to a pre-existing file on disk may have unpredictable results. This is especially true if a short file is input to replace a large file.

The CLOSE Command:

When work on a file is completed the computer must be told to CLOSE that file. If this is not done the computer will not know where to stop. (CLOSE may be used in the direct mode).

The READ Command:

This command allows us to retrieve data from a file on the disk. The computer

will look to the disk drive for all inputs of information when this command is given. Control stays with the disk drive until the CLOSE command is encountered. We must know the name and something of the structure of the file to use the READ command effectively.

To help you to understand the use of these commands in creating and reading a file it is appropriate at this time to look at an example. We will create a list of names, store it on disk and then rank the names alphabetically by use of file handling routines.

LOAD the program called TEXTER from the Systems Master Disk. Examine the program listing given below:

```
100  REM  ***** TEXTER *****
110  DIM A$(100):I = 0
120 D$ =  CHR$ (4): REM  CTRL D
130  HOME : TEXT
140  PRINT  "THIS  PROGRAM  ALLOWS  YOU  TO  STORE  LISTS":
     PRINT : PRINT "OF NAMES."
150  PRINT : PRINT "EACH LIST MUST BE GIVEN A TITLE, SUCH
     AS": PRINT : PRINT "A SUBJECT TITLE."
160  PRINT
170  PRINT :I = I + 1
180  PRINT "(PRESS THE RETURN KEY TO QUIT.)
190  PRINT "ENTER NAME No.";I
200  INPUT "";A$(I)
210  IF A$(I) <  > "" GOTO 160
220  PRINT
230  INPUT "WHAT FILE NAME? ";N$
240  PRINT D$;"OPEN ";N$
250  PRINT D$"DELETE ";N$
260  PRINT D$;"OPEN ";N$
270  PRINT D$;"WRITE ";N$
280  PRINT I - 1
290  FOR J = 1 TO I - 1
300 : PRINT A$(J)
310  NEXT J
320  PRINT D$;"CLOSE ";N$
```

This program has many significant features. The syntax used for the file handling commands should be noted, especially the use of CTRL-D. The D$ string value is defined near the beginning of the program as CTRL-D:

```
120 D$ = CHR$ (4) : REM CTRL-D
```

Whenever this function is required, simply entering PRINT D$; followed by the appropriate command in parenthesis is sufficient. For example :

```
240 PRINT D$; "OPEN "; N$
```

where N$ will represent the name chosen for your file.

Now RUN this program and see what happens. You will be able to enter any number of names up to one hundred. These names will be stored in the memory of the computer as you enter them until the RETURN key is pressed without an entry being made. You will then be asked to give the file a name. Upon entering the name you choose and pressing RETURN you will see the disk drive's 'in use' light come on for a while. If you now CATALOG the disk you will note that a new

file prefixed by the letter 'T' (for text file) has been added to the catalog. Congratulations ! You have written your first text file. Now to retrieve it in some useful manner.

LOAD the program called ALPHA from the Systems Master Disk and then examine the listing given below:

```
100  REM ***** ALPHA *****
110 D$ =  CHR$ (4)
120  HOME
130  PRINT : PRINT : PRINT : PRINT : INPUT "ENTER FILE
     NAME...";M$
140  PRINT D$;"OPEN";M$
150  PRINT D$;"READ";M$
160  PRINT : PRINT
170  PRINT "ALPHABETIZE"
180  PRINT
190  INPUT "NUMBER OF NAMES...";N: PRINT N
200  IF N = 0 THEN 410
210  DIM A$(25)
220  FOR I = 1 TO N
230  INPUT A$(I)
240  NEXT I
250  FOR I = 1 TO N
260  FOR J = 1 TO N - I
270 A$ = A$(J)
280 B$ = A$(J + 1)
290  IF A$ < B$ THEN 320
300 A$(J) = B$
310 A$(J + 1) = A$
320  NEXT J
330  NEXT I
340  PRINT
350  FOR I = 1 TO N
360  PRINT A$(I)
370  NEXT I
380  PRINT
390  PRINT D$;"CLOSE";M$
400  GET S$
410  PRINT : PRINT : PRINT : PRINT  CHR$ (4);"CATALOG"
```

Note that line 110 defines the CTRL-D character once more. This character is used in lines 140 and 150 to initiate the reading of the file from disk. At this stage the computer will look to the disk drive for all inputs until the CLOSE command is given. The keyboard has been effectively disconnected.

This brings us to an interesting feature of sequential text files: the first input from the file is not a name or any data you have entered, but the number of entries that you made in creating the files. In this case it tells you the number of names in the list. Line 190 in the ALPHA program makes use of this feature, telling us how many names to expect:

```
190 INPUT "NUMBER OF NAMES...";N: PRINT N
```

More importantly line 220 uses this information to tell the computer how many inputs to look for from this file:

```
220 FOR I = 1 TO N
```

The self documenting feature of text files has been used to effect in this program. It has many applications in file processing.

The file is read by line 230 and the loop completed by line 240:

```
230 INPUT A$(I)
240 NEXT I
```

Line 390 CLOSEs the file and returns control to the keyboard:

```
390  PRINT D$;"CLOSE";M$
```

RUN this program, using the filename you created with the TEXTER program. The file will be loaded from disk, processed and the names listed alphabetically on the screen. Data processing is really just that easy.

Another example of the use of sequential text files is included on the Systems Master disk. The program called POINTER allows you to create a co-ordinate file which can be read by the LIN REG program to give the regression co-efficients of a series of points. It is suggested that you RUN POINTER to create a file and then RUN LIN REG to read that file. The program LIN REG makes use of the MON C,I,O commands as an integral part of the program allowing you to monitor the progress of the data input. This use of the MON command can be of benefit in many applications.

The APPEND Command:

It is possible to add data to the end of an existing sequential text file by the use of this command. This is done by APPENDing then WRITEing to a file on disk. This command effectively OPENs the existing file just beyond the last field saved and allows you to commence WRITEing the new information from that point. APPEND (filename) should always be followed by a WRITE (filename).

The POSITION Command:

Data can be retrieved from or added to anywhere in a sequential text file by the use of this command. A file must first be OPENed (this sets the field position to 0 or the beginning of the file) then the POSITION command followed by the filename, a comma, the letter R then the required number, will move the address position forward that number of fields. A WRITE or READ command must follow POSITION, but remember, if you WRITE to a position in a file you write over the information that was held in that position, and that information will be lost. Great care is also needed to ensure you insert exactly the same number of characters as there were originally in that field, or the file will end up 'scrambled'. It is recommended that POSITION be used only to READ from a file, not to edit that file. Editing a sequential file is best done by READing the file into the memory of the computer, making the alterations and then WRITEing the amended file back onto the disk.

Random Access Files:

The analogy used earlier in this chapter to describe the structure of random access files was that of a card file. This analogy suits the overall nature of this type of file quite well, but does not do it justice. Access to any piece of information in the file is rapid, and updating or changeing a record within a file is quite simple, as is adding to the file. This makes random access files very useful for applications such as telephone directories, personnel information, stock lists, price charts, medical records and the like. Random

Access files take up more room on the disk to store any given amount of information than do Sequential files but do provide for far easier manipulation of data.

The commands used for creating and reading random access files are essentially the same as those used for sequential text files. They differ only in that you must define for the computer certain facts about the file you are creating. These definable facts form the main subject of our discussion on random access files.

OPEN and the L parameter:

When a Random Access file is OPENed, the Length (L) of the file must be stated. By using L50, say, we would be telling the computer that each individual record will be 50 characters long. Even if we typed in only 5 characters enough space to fit the full 50 characters would be reserved on the disk. Each set of 50 characters would constitute a 'Record' and the computer would identify it by giving it a number. This means that Record 0 would start at position 1 in the file and finish at position 50. Record 1 would start at position 51 and finish at position 100. Obviously, much thought must be given to the length chosen for each record when a file is designed. Choosing a length much greater than is strictly necessary would be very wasteful of disk space.

Note that Random Access files are not self-documenting. This means that you must record for yourself the Length of record of any file you create. Supplying an incorrect L parameter when a file is read would result in incorrect or even scrambled retrieval of data.

WRITE, READ and the R Parameter:

R when used in conjunction with a WRITE or READ command can be thought of as standing for Record Number. For instance, WRITE (filename),R1 would write record number 1 into its own reserved area on the disk. Even if we have hundreds of records on the disk, we would be able to retrieve and re-write that particular record without the risk of affecting any of the other records.

It is time now to look at an example of Random Access file creation and retrieval. LOAD the program called BABY FILER from the Systems Master disk. Examine the listing of this program:

```
100  REM  **** BABY FILER ****
110  TEXT : HOME
115  DIM B$(50)
116  DIM C$(50)
120 I = 0
130 D$ =  CHR$ (4): REM  CTRL-D
140  PRINT " This is a simple example of a program":
     PRINT : PRINT "which creates RANDOM ACCESS FILES"
150  PRINT : PRINT "You are asked to input NAMES of
     ANIMALS": PRINT : PRINT "first, followed by the name
     given to ": PRINT : PRINT "their YOUNG."
160  PRINT : PRINT "Are you ready to begin?": PRINT :
     PRINT "(enter 'Y' when you are ready)"
170  GET A$
180  IF A$ = "Y" THEN 230
190  GOTO 110
230 I = I + 1
240  PRINT : PRINT "What is animal No.";I;"?"
```

```
250  PRINT : INPUT "";B$(I)
260  IF B$(I) = "" GOTO 360
270  IF  LEN (B$(I)) > 12 THEN  GOTO 370
280  PRINT  :  PRINT  "and  what  is  a  baby  ";B$(I);"
     called?": PRINT : INPUT C$(I)
290  IF  LEN (C$(I)) > 12 THEN  GOTO 390
300  PRINT D$;"MONC,I,O"
310  PRINT D$;"OPEN NAPPY,L25"
320  PRINT D$;"WRITE NAPPY,R";I
330  PRINT B$(I): PRINT C$(I)
340  PRINT D$;"CLOSE NAPPY"
350  GOTO 230
360  PRINT D$;"CLOSE NAPPY": END
370  PRINT : PRINT "That name is too long!": FOR A = 1 TO
     2000: NEXT A
380  HOME : PRINT : PRINT : PRINT :I = I - 1:: GOTO 230
390  PRINT : PRINT "That name is too long!": FOR A = 1 TO
     2000: NEXT A
400  HOME : PRINT : PRINT : PRINT : GOTO 280
```

This program represents a simple example of Random Access file creation. It allows you to create a file called NAPPY which will contain the names of animals and what their young are called. The essential elements of this program which you should note are:

Line 230 operates the Record counter. Each time the program returns to this line the number for the next record is created.

Lines 240 to 290. These lines look for information from the keyboard and store the two inputs as string variables in this case separated by a carriage return. Note that it was necessary to DIMension these variables in lines 115 and 116 and that a limit of 12 characters has been put on the length of each of them (lines 270 and 290).

Lines 300 to 340 are the file writing routine. MON C,I,O has been put into effect to allow you to observe the functioning of this routine.

Line 310 OPENs a file called NAPPY in which each record is twenty-five characters long.

Line 320 WRITEs into the file called NAPPY the record, the number of which is defined by the value of I following the R parameter.

Line 330. This line tells the computer what data to store in the file.

Line 340 CLOSEs the file.

Line 350 returns the operation of the program back to the Record counter at line 230.

RUN this program to see how these routines operate. A text file called NAPPY will have been created on the disk. This file may contain up to 50 records each 25 characters in length.

Now LOAD the program called BABY READER and examine the listing given below:

```
100  REM  **** BABY READER ****
110  ONERR  GOTO 320
111  DIM B$(50)
112  DIM C$(50)
120  TEXT : HOME
130 D$ =  CHR$ (4): REM  CTRL-D
135  PRINT D$;"NOMON C,I,O"
140  PRINT  :  PRINT  "This  program  reads  the  RANDOM
     ACCESS": PRINT : PRINT "file called 'NAPPY' and
     answers your": PRINT : PRINT "questions about the
     names of baby ": PRINT : PRINT "animals."
145  PRINT D$;"MON I"
150  PRINT  :  PRINT "Are you ready to begin?": PRINT :
     PRINT "(enter 'Y' when you are ready)
160  GET A$
170  IF A$ = "Y" THEN 190
175  IF A$ = "y" THEN 190
180  GOTO 120
190  PRINT D$;"MON I": PRINT : PRINT : PRINT : PRINT :
     PRINT : INPUT "What animal had the baby? ";B$
200  IF B$ = "" THEN  HOME : END
210 I = 0
220  PRINT D$;"NOMON I"
230  PRINT D$;"OPEN NAPPY,L25"
240 I = I + 1
250  PRINT D$;"READ NAPPY,R";I
260  INPUT B$(I)
270  INPUT C$(I)
280  IF B$ = B$(I) THEN 300
290  GOTO 240
300  PRINT D$;"CLOSE NAPPY"
310  PRINT : PRINT "A baby ";B$(I);" is called a ";C$(I)
315  GOTO 190
320  PRINT D$;"CLOSE NAPPY"
330  PRINT "You didn't tell me that one.": FOR A = 1 TO
     1000: NEXT A
340  GOTO 190
```

This program will READ the file you have created and compare the input from that file with a keyboard input. If a match is found the file is CLOSEd and the contents of the appropriate record are displayed on the screen. If no match is found anywhere in the file the error condition resulting from reaching the end of the data causes the program to default to a predetermined message. (The ONERR statement in line 110 directs the program to line 460).

Lines 220 to 300 constitute the READ routine with lines 240 to 290 determining the number of records examined and comparing the inputs.

RUN this program and see how fast information is accessed.

No doubt by now you have thought of a good use for this type of program but you would like to be able to add to and edit the file at will. An example of how to do this is seen in the program called ANIMAL BABIES. It is suggested that you run this program and examine its listing to aid you in understanding the use of Random Access files. The various sections are clearly labelled with REM statements. A brief discussion of some points to note follows.

The Edit routine at lines 530 to 570 operates by not advancing the value of I by an increment, then accepting the re-defined value for the string B$(I) and putting this into the WRITE subroutine:

```
530  REM **** EDIT ROUTINE
540  PRINT : PRINT "What is the ANIMAL? "
550  PRINT : INPUT B$(I)
560  GOSUB 630
570  GOTO 400
```

This results in the record you decided to change being completely re-written, without affecting any other record in the file.

The UPDATE routine at lines 470 to 520 is the destination of the ONERR message at line 110:

```
470  REM  **** UPDATE ROUTINE
480  PRINT "You haven't told me that one yet.": FOR A = 1
     TO 1000: NEXT A
490  IF I = 50 GOTO 790
500 B$(I) = B$
510  GOSUB 630
520  GOTO 400
```

The last value of I to be used by the computer (the one that caused the error by making the computer look for a non-existent record, one beyond the last real record in the file) is remembered and used in the B$(I) string. This causes the next record to be written starting just beyond the end of the previous last record. Line 490 determines what happens when the limit of our DIMensioning (lines 120 and 130) is reached. If some default such as this was not defined, the error condition which would occur would result in the program cycling back to Line 460 in a continuous loop.

The B Parameter:

The file that was created by the BABY READER progam consisted of records each 25 characters long. There were no divisions within that record, the two inputs being separated simply by carriage returns. This limits our ability to search for information in each record. It is possible to define a number of fields within each record, each field being of a specified length. The B parameter performs this function. You can WRITE or READ to or from any field within any record by specifying firstly the Record (R) number followed by the address position of the desired field within the record quoted as a numeric value following the letter B. For example;

```
300 PRINT D$; "WRITE NAPPY, R";I;",B";12
```

This line would WRITE information into a field commencing 12 characters from the beginning of the Ith. record of the file NAPPY.

Remember that the computer will believe what you tell it when it comes to reading Random Access files, so you must document for yourself the structure you decided on when creating the file in order to be able to read that file effectively.

EXEC Files:

It is possible to create a type of text file which can, in itself, contain commands which can control the functioning of the computer. This is distinct from a 'computer program' written in a computer language such as basic. The text file may be used to replace the keyboard as an input device, with commands such as RUN, LOAD and SAVE being given from within the file, as well as responses to other programs requireing INPUTs.

Yes, well, now that all that has been said, let's try and make some sense of it by examining an example.

If we wished to show someone how the BABY READER program worked, we could RUN that program and supply the required INPUTs (animal names) ourselves in the usual way, from the keyboard. If we did not want to be bothered with such a demonstration, we could create an EXEC file which would RUN the desired program and then supply a number of animal names to that program for it to match. An example of this type of file is included on the Systems Master disk. It has been given the name EXECUTION.

To activate an EXEC file the command EXEC followed by the filename should be entered. Try typing EXEC EXECUTION, press RETURN then sit back and enjoy the show.

The creation of an EXEC file requires the use of the same commands as for Sequential text files. In effect, you are creating a sequential text file which will be EXECuted instead of OPENed and read. If you CATALOG the disk you will see that the letter 'T' precedes the name EXECUTION, indicating that it is a text file.

The following is a listing of a program called EXEC CREATE. This is the program which was used to write the EXEC file EXECUTION:

```
20 D$ =  CHR$ (4): REM  CTRL D
30 Q$ =  CHR$ (34): REM  "
40  PRINT D$;"OPEN EXECUTION"
50  PRINT D$;"DELETE EXECUTION"
60  PRINT D$;"OPEN EXECUTION"
70  PRINT D$;"WRITE EXECUTION"
110  PRINT "RUN BABY READER"
120  PRINT "YCAT
130  PRINT "DOG"
150  PRINT "ELEPHANT"
170  PRINT "DOLPHIN"
190  PRINT "GNU"
200  PRINT "GIRAFFE"
210  PRINT
220  PRINT D$;"CLOSE EXECUTION"
```

Note that the commands and inputs following the PRINT statements do not contain any CTRL-D or other DOS modifyer. The contents of the inverted commas are entered just as they would be from the keyboard, if you were RUNning the program BABY READER. It is obvious that you must be very familiar with the structure of the target program (in this case BABY READER) before you can make use of an EXEC file in this way.

Exec files may be used as copy programs, first capturing a BASIC program into an EXEC file, then using that file to SAVE the program to disk. It is also

possible to use EXEC files to transfer programs between languages, the most successful translation being from machine code to BASIC.

A useful example of an EXEC creation application is the program called EXEC LISTER included on the Systems Master disk. This program may be merged with any EC-BASIC program with linenumbers 10 and above, and then RUN to create a sequential text file which may be read and edited by a word processing program. The modified file can then be EXECed back into memory, and SAVEd as a program.

If this section on EXEC files has had the effect of numbing your brain, don't worry, because EXEC files are not widely used and you can live happily ever after without them. But they offer a challenge to the imagination far in excess of that offered by any other part of computing.

CHAPTER 9

S U M M A R Y o f D O S C O M M A N D S

Definitions and Abbreviations
- Filename
- Slot Number
- Drive Number
- Volume Number
- Default Values

EC-DOS Commands
- System and EC-BASIC commands
  Listing, Defaults and Syntax

Text File Commands
(i) Sequential Text Files
(ii) Random Access Files
(iii) EXEC Files

Machine Language Files

This summary is intended as a ready reference for the syntax and use of the DOS commands. For explanations of the functions of these commands, refer to the text of the preceding two chapters.

Definitions and Abbreviations of Terms Used:

FILENAME (fn):

This is the name by which DOS has been told to identify a program or data file. A valid filename is from 1 to 30 characters in length, must begin with a letter (not a number or a punctuation mark) and must not contain a comma anywhere within the name.

SLOT NUMBER (Sn):

The slot number refers to the reference number given to the various connection slots provided on the motherboard (see Chapter 3). Traditionally certain peripheral devices are connected via certain slots. The interface for the disk drives is located in Slot 6 (note that the interface forms part of the motherboard).

DRIVE NUMBER (Dn):

It is possible to specify which of your disk drives you wish to access. The drive which becomes active when you press 6 from the SYSTEM RESET menu is defined as Drive 1.

VOLUME NUMBER (Vn):

An identification number (between 1 and 254) may be given to any disk when it is INITialised. This number may then be quoted with most DOS commands. If a volume number is not specified DOS assumes the default value of 254 applies. You can find out the volume number of any disk by CATALOGing it. The volume number statement is rarely used.

DEFAULT VALUES:

The term DEFAULT VALUE means the value given by the DOS itself to certain parameters in the absence of any information about the values of those parameters being supplied by the user. In the following text any parameter enclosed in square brackets [ ] will have a default value supplied for it if you do not supply one. This would make them optional arguments, in other words you may supply the information if you wish, but if you do not the DOS supplies its own values.

EC-DOS COMMANDS

SYSTEM and EC-BASIC COMMANDS:

CATALOG [,Sn][,Dn]

Causes the names of the files present on the disk in Drive n, connected to Slot n to be displayed. For example, CATALOG,S6,D2 would cause Drive 2 which is connected to Slot 6 to be displayed. It is not necessary to use the Sn parameter if the disk drives are connected in the standard way as a default value of 6 is assumed. So to CATALOG a disk in Drive 2 all you need type is CATALOG, D2. Remember, though, that any further commands which access the disk will assume Drive 2 is the one wanted until you tell DOS otherwise. Normally the default value of Dn is 1 so that entering CATALOG would result in Drive 1 being activated.

LOAD fn [,Sn][,Dn][,Vn]

This is the command used to LOAD a program written in EC-BASIC from a disk. Note that the filename is not optional, but the other arguments are. Default values are the same as for CATALOG, for example entering LOAD HELLO would normally assume Slot 6, Drive 1, Volume 254. Entering LOAD HELLO, D2 would result in the computer looking for the program HELLO on a disk in the second disk drive and would assume Slot 6, Volume 254 as default values.

SAVE fn [,Sn][,Dn][,Vn]

This is the command used to SAVE a program written in EC-BASIC to a disk. The same constraints apply here as they do with the LOAD command. Note that if no file of the chosen filename exists on the disk SAVE will create a new file with that name, but if a file of that name does exist it will be written over unless it is LOCKed. There are also some constraints on your choice of filename. It must be from 1 to 30 characters long, must begin with a letter not a numeral or a punctuation mark and must not contain a comma. A mixture of letters numerals and spaces may form a valid filename.

RUN fn [,Sn][,Dn][,Vn]

When used in conjunction with a filename this command causes the EC-BASIC program named to be LOADed into the computer's memory and then executed. If no filename is used the program already in memory will be executed. Default values apply as with the LOAD command.

INIT fn [,Sn][,Dn][,Vn]

This command formats a new disk (or totally erases and then formats a used disk). The catalog track is prepared and the HELLO program is also saved. This is the only command which is capable of supplying a volume number to the disk. Note that the EC-DOS Systems Master disk must have been booted before this command is used. It must be followed by CALL 4096 to complete the INITialization.

DELETE fn [,Sn][,Dn][,Vn]

This command is used to remove unwanted files from the disk catalog. It will only remove UNLOCKed files and does not clear the sectors of the disk where the program is actually stored. However because the filename has been removed from the catalog the program can no longer be RUN or LOADed.

LOCK fn [,Sn][,Dn][,Vn]

This command is used to give protection to a file. A LOCKed file cannot be overwritten by a SAVE command, altered by a RENAME command or removed by a DELETE command. A file which has been LOCKed is preceded by an asterisk in the screen catalog.

UNLOCK fn [,Sn][,Dn][,Vn]

This is the logical opposite to the LOCK command. It makes the file accessible to the SAVE, RENAME and DELETE commands.

RENAME fn1,fn2 [,Sn][,Dn][,Vn]

This command allows you to alter the name by which the file is identified on the disk. For example RENAME HELLO, BONJOUR would result in the filename HELLO being replaced in the catalog by BONJOUR. The same constraints on the choice of filename apply with this command as apply with the SAVE command.

VERIFY fn [,SN][,Dn][,Vn]

This command is used to check that a particular file is stored correctly on the disk. It does this by reading the file. It is a good practice to VERIFY all of your files as you SAVE them.

PR#n

On receipt of this command all output from the computer is sent to the device connected to Slot n. For example PR#1 would send all output via Slot 1 to a printer if one was connected.

IN#n

This causes the computer to look for information to be input from a device connected to Slot n. For example if a measuring instrument was connected to the computer via Slot 7, the computer could be told to read the results coming from that instrument by the command IN#7.

TEXT FILE COMMANDS:

MON [,C][,I][,O]

Communication of both commands and information between the disk and the memory of the computer may be selectively MONitored by use of this command.

NOMON [,C][,I][,O]

This switches off the MONitoring of information flow started by the MON command.

(i) Sequential Text File Commands:

OPEN fn [,Sn][,Dn][,Vn]

Used as the first command in either WRITEing or READing sequential text files. It causes a file with the given filename to have space allocated for it in the file buffer area of memory and places the filename in the directory of the disk. Note that no length parameter is needed when this type of file is OPENed.

WRITE fn[,Bn]

Characters contained in any PRINT statements following a WRITE command will be written into the file specified by the filename quoted. If the Bn parameter is quoted the command WRITEs the information starting n positions (bytes) from the beginning of the file. Any DOS command will cancel a WRITE command.

READ fn[,Bn]

Any statements requiring an input (INPUT and GET) which follow this command will obtain the required input from the file specified with the command. Each INPUT will READ one field of the file, i.e. until a carriage return is encountered. The Bn parameter operates as for the WRITE command.

CLOSE [fn]

This is the logical opposite to the OPEN command. If used without the filename it eliminates all file buffers allocated to textfiles (except for EXEC files). If a filename is quoted that file only is CLOSEd.

APPEND fn [,Sn][,Dn][,Vn]
This command OPENs the file whose name is quoted but the information following the WRITE command is added to the end of the file. Note that APPEND should always be followed by a WRITE command.

POSITION fn,Rn
This command moves the address position of any subsequent READ or WRITE command in a previously OPENed file the number of characters specified by the R parameter.

(ii) Random Access File Commands:

OPEN fn, Ln [,Sn][,Dn][,Vn]
Used as the first command in either WRITEing or READing random access files. It causes a file with the given filename and of a given length of record (Ln) to have space allocated for it in the file buffer. It places the filename into the disk directory. Remember to record for yourself the value used with the L parameter as this must be quoted every time that file is OPENed.

WRITE fn[,Rn][,Bn]
Characters contained in any PRINT statements will be stored in the random access file whose name is quoted. If the R parameter is used the WRITE begins at the first position of the record number specified by R. If the B parameter is used the WRITE begins at the number of characters specified after the B command from the beginning of the record specified by R.

READ fn[,Rn][,Bn]
Any statements requiring an input (INPUT and GET) which follow this command will take their inputs from the file specified instead of from the keyboard. The record read can be specified by the R command and the position within that record which is to be read can be specified by the B command.

CLOSE [fn][,Sn][,Dn][,Vn] This command operates as for sequential text files.

(iii) EXEC FILES:
These files require the OPEN and WRITE commands as specified for sequential text files. To operate them the EXEC command is given. The format is as follows:
EXEC fn[,Rn][,Sn][,Dn][,Vn]

MACHINE-LANGUAGE FILES:

BSAVE fn,An,Ln [,Sn][,Dn][,Vn]
This command is used to store machine language or Binary files to disk. As well as the filename, the starting address and length (in bytes) are necessary.

BLOAD fn[,An][,Sn][,Dn][,Vn]
This command performs a LOAD on a Binary file. If no address is given with the command the same address that was given when the file was saved will be used.

BRUN fn[,An][,Sn][,Dn][,Vn]
This performs a LOAD and RUN on a Binary file. Again the address position is optional.

CHAPTER 10

G R A P H I C S

LO-RES Graphics

- GR
- COLOR
- HLIN / FROM / TO
- VLIN / FROM / TO
- PLOT
- TEXT:HOME

Example 1. LO-RES Program

HI-RES Graphics

- HGR
- HGR2
- HCOLOR
- HPLOT
- HPLOT / TO / TO
- TEXT

Example 2. HI-RES Program

The Shape of Things to Come

Shape Tables

Other Approaches

The ability to use graphics with your programs opens the door to a number of exciting possibilities. What business program does not benefit from the incorporation of graphical representation of figures? What game is not enhanced by a vivid pictorial display? Graphics allow us to accomplish all these things and more.

The EC-64 has three graphics modes in addition to the text mode we have used so far. It has a low resolution (lo res) graphics mode and two high resolution (hi-res) graphics modes.

LO-RES Graphics:

The lo res mode has a resolution of 40x40 picture elements (pixels) with the provision of 4 lines of text which comprise a text window below the graphics area of the screen. By switching on these pixels we can plot points, draw lines and outline shapes. All pixels are numbered from the origin which is the top left of the screen and is identified as 0,0. Thus the lo res screen has 40 columns across numbered from 0 to 39 and 40 rows down also numbered from 0 to 39. Attempting to light up a pixel outside these limits results in an ILLEGAL QUANTITY ERROR.

We enter the lo res mode via the BASIC command GR. Once GR is executed the screen will appear blank except for the four text lines at the bottom in which the cursor now appears. GR also causes COLOR to default to zero (black). Full screen graphics (40x48) may be implemented by poking either location -16302 or its equivalent 49234 with zero.

If your computer has a colour monitor or colour T.V. attached and the appropriate peripheral colour card installed, you will be able to display sixteen colours on the lo res screen. If you have a monochrome monitor, these colours will appear as shades of grey. The colour is set with the BASIC command COLOR = aexp where the aexp may be any value from 0 to 15. The aexp codes for the colours as follows:-

NUMBER	COLOUR
0	BLACK
1	MAGENTA
2	DARK BLUE
3	PURPLE
4	DARK GREEN
5	GREY
6	MEDIUM BLUE
7	LIGHT BLUE
8	BROWN
9	ORANGE
10	GREY
11	PINK
12	GREEN
13	YELLOW
14	AQUA
15	WHITE

Remember, that to light up a pixel, COLOR must be set to something other than black which is the default colour setting in force after GR is executed.

To specify which pixels we want lighted on the lo res screen we use the BASIC command PLOT X,Y where X indicates the column number across the screen (i.e. X co-ordinate) and Y represents the row number down the screen (i.e. Y

co-ordinate). The PLOT command lights one pixel at a time. The command SCRN X,Y allows us to determine the colour code of the pixel X,Y.

Two commands exist for drawing lines in lo res graphics. They are HLIN and VLIN which produce horizontal and vertical lines respectively. HLIN X1,X2 AT Y lights the pixels of columns X1 to X2 in row Y thus giving a horizontal line. VLIN Y1,Y2 AT X draws a vertical line from row Y1 to row Y2 at column X.

Just to summarize what we have covered so far, let's look at a short demonstration program:

```
 10   GR : REM SETS SCREEN TO LOW RESOLUTION GRAPHICS.
 20   COLOR = 13 : REM YELLOW
 30   PLOT 8,3
 40   HLIN 7,9 AT 4
 50   HLIN 7,9 AT 5
 60   COLOR= 8 : REM BROWN
 70   VLIN 6,39 AT 8 : REM POLE
 80   COLOR= 6 : REM MEDIUM BLUE
 90   FOR Y= 9 TO 35
100   HLIN 9,39 AT Y
110   NEXT : REM BLUE BACKGROUND
120   COLOR = 15 :REM WHITE
130   X = 23
140   HLIN X,X+2 AT 11
150   HLIN 10,12 AT X-1
160   HLIN 36,38 AT X-1
170   HLIN X,X+2 AT 33
180   VLIN X-2,X AT 11
190   VLIN 10,12 AT X+1
200   VLIN 32,34 AT X+1
210   VLIN X-2,X AT 37
220   REM DRAW CROSS
230   FOR I = 1 TO 3
240   HLIN 14,34 AT 20 +I
250   VLIN 14,30 AT 22 +I
260   NEXT I
280   REM OUTLINE MIDDLE STAR
290   COLOR =6
300   PLOT X,X-2
310   PLOT X,X
320   PLOT X+2,X-2
330   PLOT X+2,X
```

You'll find this program on the Tutorial/Systems Master disk that came with your computer. Type 'RUN LORES FLAG' and provided you have the disk correctly loaded into the disk drive, you'll see a lo res representation of the Eureka flag.

Now if you issue the command, TEXT, you'll see that same lo res data interpreted as text. Type in HOME to clear the screen.

HI-RES Graphics:

Both high resolution modes have the same resolution (280x192 pixels) but draw their information to be displayed from different areas of memory. The high resolution mode entered by the BASIC command HGR derives its display

information from page 1 of memory (8k -16k). The other hi-res mode entered via the command HGR2 displays information from page 2 (16k -24k) of memory.

HGR:

When HGR is executed a hi-res graphics screen (280 x160) is set up with a four line text window. Note that text screen memory is untouched and is still at full screen although the top 20 lines are overlayed with the high resolution graphics screen. Thus the cursor will only be visible if it is on the last four lines of the text screen. The hi-res graphics may be set to full screen by poking -16302 or its equivalent 49234 with zero.

Programs which eat up great chunks of memory may be prevented from writing to page 1 of memory by setting HIMEM equal to 8192.

HGR2:

Executing HGR2 enables a full screen hi-res graphics mode of 280x192 pixels. No text memory is utilized and since page 2 of memory provides the information to be displayed, more memory is made available to the programmer than would otherwise be the case if HGR page 1 had been used instead. Poking zero into address -16301 allows the use of a text window on the bottom four lines. However, those four lines of text are drawn from page two of memory and are therefore difficult to utilize.

Setting HIMEM equal to 16384 will render your page 2 graphics data safe from your BASIC program.

Colour is set in a similar fashion to that in the lo res mode although only 8 colours are available. The BASIC command is HCOLOR = aexp and the colours are as follows:

aexp	COLOUR
0	black
1	green
2	blue
3	white
4	black
5	green
6	blue
7	white

Owing to the way in which colour T.V.s produce colours, the colours actually shown on the screen may vary from those described. Points plotted on the hi-res T.V. screen using the same colour will vary depending on whether the column in which the point is plotted is even or odd. The colour of the pixel will also depend on whether two pixels of adjacent columns are lit. Thus three different colours may be obtained using the same value of HCOLOR depending on the position of the point plotted. This is known as colour artifacting and advanced programmers actually employ this phenomenon to create special graphics effects.

HPLOT is the BASIC command used to plot a point on either of the two hi-res graphics screens. It may be used in the same way as PLOT is used in the lo res mode e.g. HPLOT 150,90 where 150 and 90 specify the x and y co-ordinates respectively. However, HPLOT is far more versatile than PLOT and can be used to draw a straight line from one point to another. HPLOT TO X1,Y1 will draw a line connecting the last previously plotted point to point X1,Y1. HPLOT X1,Y1 TO

X2,Y2 TO X3,Y3 TO X1,Y1 will draw straight lines connecting the three points specified giving a triangle. Try the following program:

```
 2  REM  HIRES 1
 5  HIMEM: 16384: REM PROTECT PAGE TWO
10  HGR2: REM SET HI-RES GRAPHICS #2
20  HCOLOR= 3: REM WHITE
30  HPLOT 0,0 TO 279,0 TO 279,191 TO 0,191 TO 0,0: REM
    DRAW BORDER
40  HCOLOR= INT (RND (1) *8) : REM CHOOSE RANDOM COLOUR
    FROM 0 TO 7
45  REM DRAW RANDOM SIZE RECTANGLE AT A RANDOM POSITION
    ON THE SCREEN
50  X = INT (RND (1)* 279):X1 = X+INT (RND (1)* 100)
55  IF X1> 279 THEN X1= 279: REM KEEP RECTANGLE WITHIN
    SCREEN
60  Y = INT (RND (1)* 191):Y1=Y+INT (RND (1)* 100)
65  IF Y1> 191 THEN Y1= 191: REM KEEP RECTANGLE WITHIN
    SCREEN AREA
70  HPLOT X,Y TO X1,Y TO X1,Y1 TO X,Y1 TO X,Y
75  REM FILL RECTANGLE WITH COLOUR
80  FOR I = 1 TO Y1-Y
100 HPLOT X.Y +I TO X1,Y +I
110 NEXT
120 GOTO 40: REM DRAW ANOTHER RECTANGLE
```

To regain control of your computer after running these graphics programs, type CTRL-C and then TEXT followed by HOME.

The Shape of Things to Come:

EC-BASIC has the ability to handle previously defined shapes on the high resolution graphics screen. The BASIC commands which accomplish this task are DRAW, XDRAW, ROT, SCALE and SHLOAD.

The shape of the object is defined as coded instructions. This code, which is stored as a series of bytes in a predetermined area of memory, instructs the computer to plot a particular point and move in a certain direction. Four directions are possible, up, down, left and right. Of course, the point may or may not be plotted as so desired.

Each byte of code contains three separate instructions:

ONE BYTE							
3RD INSTRUCTION		2ND INSTRUCTION			1ST INSTRUCTION		
7	6	5	4	3	2	1	0
MOVEMENT		PLOT	MOVEMENT		PLOT	MOVEMENT	

The first and second instructions specify plotting and movement information while the third instruction contains only movement information. This information is specified as in the following table:

INSTRUCTION	BIT	INFORMATION
	0	MOVEMENT
1	1	
	2	PLOT
2	3	MOVEMENT
	4	
	5	PLOT
3	6	MOVEMENT
	7	

The bit pairs which contain movement information code for specific directions as follows:

VALUE OF BIT PAIR (BINARY)	DIRECTION
00	UP
01	RIGHT
10	DOWN
11	LEFT

If a plot bit = 1, a point is plotted. If the plot bit = 0, the point is not plotted. When the instruction is executed, plotting occurs first followed by movement. The computer evaluates the shape byte from the least significant to the most significant bit i.e. from bit 0 to bit 7. If either the first or second instructions are followed by instructions or an instruction which is equal to zero then the instructions following it are ignored. Thus the byte may not end with a third instruction coded for upward movement as this evaluates to zero and will be ignored by the computer. Likewise, the first instruction cannot be followed by a second instruction coded for upward movement and no plot unless the third instruction is coded for movement in a direction other than up. A byte which contains eight zeros signifies the end of the shape definition.

Shape Tables:

The easiest way to compose a shape table is to draw the shape on graph paper, decide from which point you want to start plotting and translate the plotting and movement data into coded bytes. As an example, let's compose a shape table for an arrow.

Only pixels 1,2,3,4,5,6,9,11,13,14,15,17,19,21 and 23 are plotted to give an arrow graphic. (See the diagram on the next page).

Starting at position one we number the movements we will make in plotting our shape. The points to be plotted are shaded. Now let us translate these plots and movements into coded bytes. Instructions one and two of byte zero both plot and move up. The next movement is plot and move up but we cannot plot with instruction three so we skip it and use instruction one of the next byte to code for this. Byte eleven tells the computer that it has reached the end of our shape definition. (See the table on the next page).

GRAPHIC ARROW

```
          16->17->18
          15<-14  19->20
      10->11->12->13      21->22
       9<--8<--7<--6          23
                5
                4
                3
                2
                1
```

	INSTRUCTION		
BYTE	3	2	1
0		100	100
1		100	100
2		011	111
3	01	100	011
4		001	101
5		111	100
6		001	100
7		010	101
8		010	101
9		010	101
10			101
11	000	000	000
	(END OF SHAPE DATA)		

	BINARY BYTES INSTRUCTION			
BYTE	3	2	1	HEX BYTES
0	00	10 0	100	24
1	00	10 0	100	24
2	00	01 1	111	1F
3	01	10 0	011	63
4	00	00 1	101	0D
5	00	11 1	100	3C
6	00	00 1	100	0C
7	00	01 0	101	15
8	00	01 0	101	15
9	00	01 0	101	15
10	00	00 0	101	05
11	00	00 0	000	00
		!		
	1st HEX DIGIT - 2nd HEX DIGIT			

CONVERSION TABLE

DECIMAL	HEXADECIMAL	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Having coded our shape definition in binary code we must now translate it into hexadecimal code so that we will be able to enter it into the computer. Each byte containing three instructions is translated into a two digit hexadecimal number. Note that the skipped instructions are filled up with zeros.

A byte in hexadecimal is represented by a two-digit number. The first digit of the hex byte represents bits four, five, six and seven of the binary byte. The second digit represents bits zero, one, two and three. Incidentally, four bits or half a byte is sometimes termed a nybble.

Once we have translated our shape definition into a series of hexadecimal bytes we have only to provide an index to accompany it and we have completed our task. The index heads the shape table of shape definitions. The first byte of the index contains the number of shape definitions in the table (coded in hex 0-255). The next byte is unused. The third and fourth bytes contain the location of the start of shape definition #1 in relation to the starting address of the shape table. The location is written with the two least significant hex digits comprising the third byte and the two most significant digits of the location comprising the fourth byte. The location of the second shape definition would be written in the fifth and sixth bytes and so on up to a maximum of 255 shapes. Since we have defined only the one shape we can place it immediately below the index so the location of our shape would be written in the third and fourth bytes as 00, the most significant byte (MSB), 04, the least significant byte (LSB). As the least significant byte is recorded first the third byte would be written as 04 and the fourth byte as 00. The fifth byte in this example is the start of our shape definition.

Our Shape table is recorded as in the diagrams on the following page.

To enter our shape table into the computer we must choose as our start address the highest memory address which will not be wiped out by the command HGR or HGR2. The address 1DF0 is just below HGR page one of memory and should be safe for our purposes. Call up the EC-64 monitor by typing CALL-151 from BASIC and open the start address by typing a colon immediately after the address (1DF0:). Now type in the hex numbers (one byte) at a time in order, separated by spaces and hit return.

LOCATION	
1DF0 = START ADDRESS	+ 0 BYTES
1DF1 = ..	+ 1 ..
1DF2 = ..	+ 2 ..
1DF3 = ..	+ 3 ..
1DF4 = ..	+ 4 ..
1DF5 = ..	+ 5 ..
1DF6 = ..	+ 6 ..
1DF7 = ..	+ 7 ..
1DF8 = ..	+ 8 ..
1DF9 = ..	+ 9 ..
1DFA = ..	+ 10 ..
1DFB = ..	+ 11 ..
1DFC = ..	+ 12 ..
1DFD = ..	+ 13 ..
1DFE = ..	+ 14 ..
1DFF = ..	+ 15 ..

CONTENTS		
1DF0 = 01 = NUMBER OF SHAPES		SHAPE
1DF1 = 00 = UNUSED		TABLE
1DF2 = 04 = LSB	LOCATION OF	INDEX
1DF3 = 00 = MSB	SHAPE #1	
1DF4 = 24		
1DF5 = 24		
1DF6 = 1F		
1DF7 = 63		
1DF8 = 0D		
1DF9 = 3C	SHAPE DEFINITION # 1	
1DFA = 0C		
1DFB = 15		
1DFC = 15		
1DFD = 15		
1DFE = 05		
1DFF = 00 = END OF SHAPE DEFINITION		

EC-BASIC expects to find the shape table's start address in memory location E8 (LSB) and E9 (MSB) so we will store 1DF0 in these locations by typing E8:F0 1D and pressing return.

Setting HIMEM to the start address of the shape table will prevent unexpected disasters so type 73:F0 1D

[You may save the shape table on tape by entering the length of the table in locations 0 and 1 in the usual format (LSB,MSB) and then writing the table itself from start address to end address to tape. You can accomplish this in our example by typing 0:0F 00 (table length)

0.1W 1DF0.1DFF

If you have your cassette player ready to record, press return and the shape table will be stored on tape. To load this data from EC BASIC type SHLOAD and press return when the tape recorder is ready to play.]

Now that you have entered a shape table into the computer, you can manipulate it using a BASIC program.

```
 10  REM  HIRES SHAPE
 20  HGR
 30  HCOLOR= 3
 40  ROT= 0
 50  SCALE= 1
 55  DRAW 1 AT 139,80
 60  FOR R= 0 TO 64
 65  DRAW 1 AT 139,80
 66  FOR I = 1 TO 20: NEXT I
 68  XDRAW 1 AT 139,80
 70  ROT= R
 75  XDRAW 1 AT 139,80
 80  NEXT R
 90  FOR S = 1 TO 11
 92  DRAW 1 AT 139,80
 94  FOR I = 1 TO 100: NEXT I
 96  XDRAW 1 AT 139,80
100  SCALE= S
110  NEXT S
120  DRAW 1 AT 139,80
130  GOTO 60
```

This simple example program illustrates the use of the commands DRAW, XDRAW, ROT and SCALE. DRAW places a specified shape from the shape table on the hi-res screen at the specified x,y position. If the position is not stated the shape is drawn at the last point HPLOTed, DRAWn or XDRAWn. XDRAW is similar to DRAW except that it DRAWs the shape in the complementary colour to that already existing at the points plotted. White is the complement of Black and Green is the complement of Blue. Using XDRAW you can rub out a shape created with DRAW by redrawing it in its complementary colour.

ROT specifies the orientation of the shape on the screen. The shape may be ROTated from its original orientation (ROT=0) clockwise through 360 degrees (ROT=64). Four rotation positions are possible if SCALE= 1.

ROT=	ORIENTATION
0	ORIGINAL
16	90 DEGREES
32	180 DEGREES
48	270 DEGREES
64	360 (ORIGINAL)

If SCALE= 2 then eight different orientations are possible and so on.

The size of the shape is determined by the SCALE command. The original defined size of the shape corresponds to SCALE= 1. Subsequent values up to 255 enlarge the original points by the number of times specified by the arithmetic expression. The points are enlarged in the direction specified by the instructions when composing your shape definition. This should be taken into account when defining your shape as enlargements of the original using the SCALE command can leave the enlarged shape bearing little resemblance to the original. We have seen an example of this when our arrow shape was SCALEd up. It is usually more satisfactory to create the shape at the upper end of the

size range required and then SCALE down from there.

Other Approaches:

A simpler way to create graphics is to make use of commercially available graphics programs such as a High Resolution Character Generator. This type of program allows the creation of character fonts which may be used to replace the normal symbols generated on the screen by keystrokes. This can be used to gain an effect or to title a program.

Another approach to graphics is to use one of the many hardware/software packages now on the market. The use of these systems simplifies the use of graphics considerably. An introduction to the fun that such an addition to your computer can provide can be gained by running the PADDLEPLOT program on your Systems Master Disk and drawing shapes using the joystick controller.

CHAPTER 11

P E R I P H E R A L S

Expansion Slots and Games port
- Conventional use of Slots

Printer
- accessing
  - PR#1
  - PR#0
- How to do a listing on the printer

Modem/Communications card

80 Column Cards

RAMcards

PAL Colorcard

Graphics Tablets

11: Peripherals

The aim of this Chapter is to acquaint the user with some of the devices which may be connected to your computer to enhance its capabilities. As there are many types of peripherals available we will only deal with the most commonly encountered ones.

Our first consideration must be to discuss how these accessories are attached to the computer system. There are two possible modes of connection; via the Games Port, or via the Slots. Refer to the Photographs in Chapter 2 to identify these structures on the motherboard of the computer.

The Slots are provided as a convenient way of expanding your computer's capabilities by the use of interface cards. These interface cards are necessary to allow the computer and the attached peripheral device to communicate with each other. Unfortunately most peripherals require some type of interface to connect them to the computer. In some instances the Games Port may be used instead. Generally devices that attach via the Games Port do not require further interfacing.

The table below may be used as a guide to the connection of peripherals. Certain devices are traditionally attached to certain slots. These traditions are perpetuated in the table:

DEVICE	CONNECTION	POSITION
RAMcards	Interface Card	Slot 0
Printer	Interface Card	Slot 1
Modem/Communication	Interface Card	Slot 2
80 Column Card	Interface Card	Slot 3
Z80-A CPU	Built-in	Slot 4
Disk Controller	Built-in	Slot 6
Speech Synthesizer	Interface Card	
PAL Colorcard	Interface Card	Slot 7
Graphics Tablets		Games Port

1. PRINTER:

A Printer is probably the most useful single addition that can be made to your computer system. Apart from being able to obtain instant print-outs of your programs, files, data and graphics, the addition of a printer allows your computer to be used as a word processor.

There are several types of printers on the market. The basic choice is between the relatively cheap and fast 'Dot Matrix' printers and the more expensive, slower, high quality 'Daisy Wheel' printers. The choice of types will depend entirely on your needs and preferences.

No matter what printer you may buy, it will require an interface card. This is a computer card which contains the necessary software to allow the computer to drive the printer. As seen from the previous table, the printer interface is traditionally inserted in Slot 1. It is advisable to observe this convention as some commercial software assumes that Slot 1 has been used.

Accessing the printer directly from the computer's keyboard is achieved by the use of the PR#1 command (assuming Slot 1). Once the printer has been activated output to the printer will continue until either a PR#0 command is given (PR#3 if in the 80-column mode) or until a CTRL-RESET has occurred.

To activate the printer from within an EC-BASIC program the PR#1 command is treated as a DOS command and is prefixed by a CTRL-D. (See the EC-DOS Chapter).

Most commercial software allows for easy access to the printer by use of commands built into the programs. The documentation accompanying these programs will provide the necessary details.

2. MODEM / COMMUNICATIONS CARD:

The use of a MOdulator/DEModulator in conjunction with a Communications Card allows you to connect your computer via a telephone line to another computer or data storage system. This aspect of computer useage is growing rapidly, and electronic publication is already a big business. Electronic mail has also come into vogue.

3. 80 COLUMN CARD:

The normal video display from the EC-64 is in the 40/24 format, that is there are forty columns (or characters) capable of being displayed across the screen and twenty-four rows displayed down the screen. If word processing is to be undertaken this format is a little unwieldy, as normal typing paper is capable of taking 80 characters across. This may make screen formatting difficult in some circumstances. To overcome this problem a computer card which is capable of generating slightly smaller characters in an 80 column format may be fitted.

Slot 3 is the usual home of such a card. Much of the CP/M System software searches this Slot to see if an 80 Column Card is fitted.

Normal EC-BASIC programs may be run in 80 column mode by accessing the card with the PR#3 command.

4. RAM CARDS:

It is possible to expand the memory of your computer beyond its standard 64K by the use of these cards. Random Access Memory is now relatively cheap and may be used to enhance the performance of your computer.

RAMcards must be inserted in Slot 0.

5. PAL COLORCARD:

If use is to be made of colour graphics a colour monitor or modified television is necessary. To make the EC-64 compatible with PAL-D colour systems a special interface card called a PAL COLORCARD is necessary. This is usually inserted in Slot 7 and connected directly to the colour monitor or to a colour television by an RF Modulator.

6. GRAPHICS TABLETS:

The use of some direct means of Graphics input has many attractions, the main one of which is ease of image creation. There are several devices on the market which are moderately priced and extremely effective in this area. Most require no expensive interfaces as they connect directly through the Games Port. They come with their own software which can turn your computer into an artist's palette.

CHAPTER 12

The EC MONITOR

The EC System Monitor:

The System Monitor program is contained within the ROM (Read Only Memory) of your computer. It allows the user to address the many memory locations within the computer directly and to examine or change the contents of those locations. The same function can be performed from BASIC using the PEEK and POKE commands, but this method is satisfactory only when a small number of memory locations are involved. The EC System Monitor program provides a far more efficient way of dealing with large numbers of memory changes.

To understand what the System Monitor is and how to use it effectively, the user must have some knowledge of the structure and organization of the computer's memory. The '64' in 'EC-64' refers to the memory capacity of the computer. It has 64 Kilobytes of memory, a 'Kilobyte' being 1,024 'bytes'. Each 'byte' of memory stores eight 'bits' of information. This information may be part of a machine language routine, the value of a variable, a character, or even a 'pointer'.

Each of the bytes is identified by a unique memory location which is referred to as its 'address'. The 'pointer' referred to above simply contains the address of the item to which it points. Thus a pointer may indicate the start of a BASIC program, the current line of a BASIC programme, the start of the variable storage area and so-on.

Using a knowledge of the structure of the memory, the programmer has this information available and is able to make use of it via the System Monitor Program. Specific memory locations may be viewed individually or sequentially and even moved to other locations. It is also possible to compare the contents of two different locations. The final step is to write programmes in Machine or Assembly Language. Although this is relatively difficult, the resultant programmes execute directly without using the EC-BASIC interpreter, making them much faster in their execution.

Entering and Leaving the EC System Monitor:

Typing 'CALL -151' or 'CALL 65385' from BASIC will call up the Monitor Program. Both of these values, when used after a 'CALL' statement, represent the starting address of this program. The visible evidence of entering the Monitor is that the prompt shown on the screen changes from] to #. To get back into BASIC all you need to do is type 'CTRL-C' or, if DOS is loaded, '3D0G' (that 0 is a zero not the letter O).

The Monitor may be entered directly from the System Reset Menu by selecting the 'M' option. However, as no DOS would have been loaded, no disk storage of any programs would be possible.

Using the Monitor Program:

All numbers entered via the Monitor, whether they be addresses or values, must all be in hexadecimal form. Each byte may be represented by two hexadecimal digits and one memory location may contain a number with a decimal value within the range 0 to 255 inclusive. As the computer's memory capacity is 64 Kilobytes (65,536 bytes) the addresses of each memory location may be represented by four hexadecimal digits or two bytes.

To examine the contents of a memory location type 'E' followed immediately by the address of the location you wish to examine. The Monitor will respond with the address you typed in followed by a dash, a space and the hexadecimal number

contained in that location. For example, to examine the contents of Memory Location 0001, enter

```
E0001
```

(Remember to include all of the zeros when entering the address of a location.)

The machine will respond with

```
0001- 3C          <
```

The characters which appear on the extreme right of the screen (in columns 33 to 39) represent the value of the location interpreted as an ASCII character. The value 3C in hexadecimal is the equivalent of the decimal value 60, which is interpreted in ASCII as <.

When the Monitor is commanded to display the contents of a memory location it stores the address of that location. The computer uses this address when a 'memory dump' is performed.

A memory dump is a display of the contents of a range of addresses (i.e. a number of memory locations). To perform a dump the first and last addresses of the range to be examined are entered, separated by a full stop. Try the following example:

```
# 0001.0011
  0001- 3C  D4  4C  3A  DB  FF  FF
  0008- 00  00  4C  99  E1  22  00  6B
  0011- 00  00
```

If a full stop followed by an address is entered, a memory dump of the contents of the memory locations whose addresses range from the last address referred to up to the address following the full stop will be displayed. Pressing RETURN displayes a single line of a memory dump beginning at the last address opened.

To change the contents of a memory location type the address of the location followed by a colon and the value you wish to be entered at that location. For example:

```
# 300:AA
```

Press RETURN to enter this into the memory.

The values of up to 85 consecutive addresses may be changed simultaneously by typing the start address, a colon, followed by the values to be entered. These valued must be separated by spaces. For example:

```
# 300:01  FA  C1  0D  EE  91  5C
```

Blocks of memory may be shifted from one area of memory to another using a 'move' command which is built into the Monitor program. This is done by entering the destination address followed by the symbol <. This must be followed by the start and end addresses of the block to be moved separated by a full stop. The letter 'M' for 'move' is then appended and the RETURN key is pressed. For example:

```
# 220 < 300.313 M
```

would move the contents of memory addresses 300 to 313 back to locations starting at 220.

To demonstrate what has been discussed so far work through the following examples. In it we will alter the contents of memory locations 300 to 313 (hexadecimal) and then move that block of memory to a series of locations starting at 220. Enter the following:

```
# 300: 1  2  3  4  5  6  7  8  9  10  11  12  13  14
       15  16  17  18  19  20
```

This has altered the contents of the 20 locations which lie between 300 and 313 hexadecimal.

To move this block enter the following:

```
# 220 < 300.313 M
```

The transfer has been performed.

To verify that the transfer has been successful enter:

```
# 220 < 300.313 V
```

The letter 'V' represents the 'verify' command. If the comparison fails the Monitor will display the addresses in the range of memory 300 to 313 whose values are different from the values in the destination range. To test this out alter address 225 by entering:

```
# 225: AA
```

Then use the verify command as before by entering:

```
# 220 < 300.313 V
```

The Monitor should respond with:

```
0305- 06  (AA)
```

The Monitor has displayed the address in the original range whose value differed, followed by the value contained in that location, as well as the value held in the corresponding destination address in brackets.

Monitor Commands:

The Monitor program includes commands which facilitate the writing of machine language programs. Some of these commands will now be introduced.

The G Command:

Entering G after an address instructs the microprocessor to begin execution of a machine language program starting at that address. The final statement in the program should be 60 which is the hexadecimal op-code for RTS (Return From Sub-routine). This returns control to the Monitor.

The L Command:

This command, when used immediately after an address, causes the Monitor to

list 20 lines of the assembly language translation of the machine op-codes and operands. The Monitor displays the addresses of the locations followed by the op-codes and operands in hexadecimal format and then the three letter assembly language mnemonic with operands as appropriate. Entering L again will result in the next 20 lines of assembly language translation being displayed.

The CTRL-E Command:

Entering CTRL-E causes the Monitor to display the 6502 microprocessor's five registers. These are:

A = Accumulator
X = X Register
Y = Y Register
P = Processor Status Register
S = Stack Pointer

The contents of the registers may be changed by entering a colon followed by the new values for the registers. The following example demonstrates this: Enter CTRL-E and press RETURN.

The Monitor should respond with:

```
A=00  X=FF  Y=FF  P=00  S=00
```

To change these values enter the following:

```
:AA  BB  CC  DD  EE
```

Now re-examine the contents of the registers using the CTRL-E command to confirm that the changes have been made.

The S Command:

This is the 'step' command. It allows the programmer to step through a machine language program one instruction at a time. This command displays each decoded instruction as it is executed. To demonstrate this command enter:

```
# 600S
```

The Monitor should respond with:

```
0600- A0  C1            LDY    #$C1
      A=00  X=00  Y=C1  P=B0  S=F4
```

The T Command:

This command activates the 'trace' function. It is very similar to the S (step) function except that it traces through the program automatically until it encounters a BRK instruction. Enter #600T and examine the result.

The I and N Commands:

These commands alter the video output to inverse (I) and back to normal (N).

The CTRL-P Command:

Precedeing CTRL-P with a Slot Number results in the activation of the interface

card in that slot. For instance, if a printer interface was in Slot 1, entering 1CTRL-P would direct the Monitor's output to the printer. Entering 0CTRL-P would return the output to the video screen.

Exiting the Monitor:

Three commands are provided that allow BASIC to be re-entered:

CTRL-B performs a BASIC COLD function, returning to the BASIC language but clearing all memory.

CTRL-C performs a BASIC WARM function, leaving any BASIC program in memory intact. Any variables in memory will also be preserved.

3DOG will perform the same function as CTRL-C if DOS has been previously loaded.

Monitor Subroutines:

There are a number of useful subroutines within the Monitor Program which may be accessed from within your own programs. A list of the start addresses of the more useful routines may be obtained from the manufacturer.

APPENDIX 1

E R R O R M E S S A G E S

(a) EC-DOS Error Messages:

The DOS program, like any computer program, expects commands to be given in a certain format. It also expects to find certain physical conditions to exist within the computer system (such as a disk in a disk drive). It expects to be able to believe you when you give it a command in a recognised format.

If the program doesn't find what it expects in any one of these areas, one of a series of error messages will be displayed. The message sent depends on the particular error DOS encounters and is designed to help you identify that error.

This section lists the various DOS error messages and gives a discussion of what they mean and what to do to correct the problem.

SYNTAX ERROR:
The format, spelling or syntax of the command was incorrect. Check your input with the formats given in the Summary of DOS Commands, Chapter 9.

I/O ERROR:
This means that an Input Output Error has occurred when the DOS has tried to read from or write to a disk in the disk drive. Check that there is, in fact, a disk in the drive, that it is inserted correctly and that the door on the drive is closed. If these conditions are met and the error still occurs, it indicates a disk error of some type. Try another disk to make sure the drive is functioning correctly.

FILE TYPE MISMATCH:
The DOS command you issued was not appropriate for the file you named. For example, if you try RUNning instead of BRUNning a B (binary) file, this message will appear. Check the type of file you are calling by CATALOGing the disk.

FILE NOT FOUND:
The filename you entered did not match any filename on the disk. Check the contents of the disk by CATALOGing it.

FILE LOCKED:
The file you tried to DELETE or SAVE to has been protected by LOCKing. If SAVEing, choose another filename to SAVE under, or UNLOCK the file if you are sure you want to replace it. If you are serious about DELETEing the file, you must UNLOCK it first.

WRITE PROTECTED:
When SAVEing or WRITEing a program, this message means that the disk in the drive has the write-protect sticker in place. Remove the sticker from the notch on the left side of the disk, and try again.

DISK FULL:
There is insufficient room on the disk to store the program you wish to SAVE. INITialize a new disk to take your program, or use another initialized disk with sectors left unused.

LANGUAGE NOT AVAILABLE:

The program you tried to RUN requires another version of BASIC. If you have that version of the language on disk, load it into memory and then try again.

PROGRAM TOO LARGE:

Not enough memory is available to hold the program. Switching off the computer and re-booting may help in some cases. Try typing FP and pressing RETURN. This re-organises the memory into the most efficient manner, making more room. (See also the HIMEM and LOMEM section in the EC-BASIC chapter).

NO BUFFERS AVAILABLE:

Too many files are in use at the same time, meaning the MAXFILES command has been set too low for the current situation. Set MAXFILES to the appropriate value. (See the MAXFILES section of Chapter B).

END OF DATA:

If this message is displayed when reading a textfile it indicates that you have tried to read from an area in the file where no data exists, or you have tried to read beyond the last record of the file. You should carefully re-evaluate the file retrieval program and examine your records regarding the structure of the file you are trying to retrieve.

NOT DIRECT COMMAND:

Seen if you try to use any of the file handling commands directly from the keyboard (i.e. in the DIRECT mode) instead of from within a program (i.e. in the DEFERRED mode).

RANGE ERROR:

Displayed if the value accompanying a command exceeds the acceptable range of values for that command. (See EC-DOS Commands, Chapter 9).

VOLUME MISMATCH:

The Volume number used with a DOS command does not agree with the Volume number of the disk in the disk drive. CATALOG the disk to check the volume number, then insert the correct disk or LIST the program to find the command which has the incorrect volume statement and correct it.

(b) EC-BASIC Error Messages:

When EC-BASIC encounters an error in a program, it responds with a brief description of the error that has occurred and the line number in which the error was found.

BAD SUBSCRIPT: An array variable has been incorrectly DIMensioned.

CAN'T CONTINUE: The computer is unable to CONTinue execution of the program.

DIVISION BY ZERO: An attempt to divide by zero was encountered.

FORMULA TOO COMPLEX: The computer is unable to evaluate the expression.

Appendix 1: Error Messages

ILLEGAL DIRECT: An attempt was made to use a command which the computer does not recognize in direct mode.

ILLEGAL QUANTITY: A BASIC function encountered a value which was out of range. This error can occur when string functions are used with an illegal argument, negative array subscripts, attempting to find the SQR of a negative value etc.

NEXT WITHOUT FOR: A NEXT statement was encountered before a FOR or did not correspond to the FOR previously encountered.

OUT OF DATA: Too few DATA items for READ statement, or a READ statement was encountered after all DATA items had been read.

OUT OF MEMORY: The computer has insufficient memory to run the program, or too many variables are used, or too many nested GOSUBs, FOR/NEXT loops or parentheses are used, or an expression was encountered which was too complex.

OVERFLOW: This message is given when the computer encounters a number which is too large to be handled by EC-BASIC.

REDIM'D ARRAY: An attempt was made to DIMension an array which had been previously DIMensioned.

RETURN WITHOUT GOSUB: A RETURN was executed before a corresponding GOSUB.

STRING TOO LONG: EC-BASIC cannot handle a string of more than 255 characters.

SYNTAX ERROR: A mistake in format, punctuation etc. of a statement was encountered.

TYPE MISMATCH: An attempt was made to assign a string to a numeric variable or vice versa.

UNDEF'D FUNCTION: The program attempted to execute a function which was not previously defined.

UNDEF'D STATEMENT: A statement directed a branch in execution to a non-existent line number.

APPENDIX 2

A S C I I C O D E S

ASCII is one of those much loved American acronyms which stands for the American Standard Code for Information Interchange. What is it all about, you ask? Well, when you press a key on the computers keyboard, an ASCII code number for that character is transmitted to the computer. The computer returns ASCII code numbers for each character to be printed on the screen or printer. You can make use of these code numbers in your BASIC programs through the CHR$ (aexp) and ASC (sexp) functions. The series of characters generated by the ASCII codes from 0 to 127 is repeated for values of 128 to 255. Attempting to use values greater than 255 with the CHR$ function will produce an Illegal Quantity Error.

CODE		ASCI		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
0	0	NUL	null	CTRL-@	
1	1	SOH	start of header	CTRL-A	
2	2	STX	start of text	CTRL-B	
3	3	ETX	end of text	CTRL-C	
4	4	ET	end of trans-mission	CTRL-D	
5	5	ENQ	enquiry	CTRL-E	
6	6	ACK	acknowledge	CTRL-F	
7	7	BEL	bell	CTRL-G	Sounds speaker
8	8	BS	backspace	CTRL-H or <-	
9	9	HT	horizontal tab	CTRL-I	
10	A	LF	line feed	CTRL-J	
11	B	VT	vertical tab	CTRL-K	
12	C	FF	form feed	CTRL-L	
13	D	CR	carriage return	CTRL-M or RETURN	
14	E	SO	shift out	CTRL-N	
15	F	SI	shift in	CTRL-O	
16	10	DLE	data link escape	CTRL-P	
17	11	DC1	direct control 1	CTRL-Q	
18	12	DC2	direct control 2	CTRL-R	
19	13	DC3	direct control 3	CTRL-S	
20	14	DC4	direct control 4	CTRL-T	
21	15	NAK	negative acknowledge	CTRL-U or ->	
22	16	SYN	synchronous	CTRL-V	
23	17	ETB	end transmission block	CTRL-W	
24	18	CAN	cancel	CTRL-X	
25	19	EM	end medium	CTRL-Y	
26	1A	SUB	substitute	CTRL-Z	
27	1B	ESC	escape	ESC	
28	1C	FS	field separator	-	
29	1D	GS	group separator	CTRL-SHIFT-M	
30	1E	RS	record separator	CTRL-^	
31	1F	US	unit separator	-	
32	20	SPACE		SPACE	

CODE		ASCII		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
33	21	!		!	!
34	22	"		"	"
35	23	#		#	#
36	24	$		$	$
37	25	%		%	%
38	26	&		&	&
39	27	'		'	'
40	28	(		(	(
41	29	)		)	)
42	2A	*		*	*
43	2B	+		+	+
44	2C	,		,	,
45	2D	-		-	-
46	2E	.		.	.
47	2F	/		/	/
48	30	0		0	0
49	31	1		1	1
50	32	2		2	2
51	33	3		3	3
52	34	4		4	4
53	35	5		5	5
54	36	6		6	6
55	37	7		7	7
56	38	8		8	8
57	39	9		9	9
58	3A	:		:	:
59	3B	;		;	;
60	3C	<		<	<
61	3D	=		=	=
62	3E	>		>	>
63	3F	?		?	?
64	40	@		@	@
65	41	A		A	A
66	42	B		B	B
67	43	C		C	C
68	44	D		D	D
69	45	E		E	E
70	46	F		F	F
71	47	G		G	G
72	48	H		H	H
73	49	I		I	I
74	4A	J		J	J
75	48	K		K	K
76	4C	L		L	L
77	4D	M		M	M
78	4E	N		N	N
79	4F	O		O	O
80	50	P		P	P
81	51	Q		Q	Q
82	52	R		R	R
83	53	S		S	S
84	54	T		T	T
85	55	U		U	U

CODE		ASCII		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
86	56	V		V	V
87	57	W		W	W
88	58	X		X	X
89	59	Y		Y	Y
90	5A	Z		Z	Z
91	5B	[		[	[
92	5C				
93	5D	]		[	]
94	5E	^			^
95	5F	_			_
96	60				
97	61	a		SHIFTLOCK-A	a
98	62	b		SHIFTLOCK-B	b
99	63	c		SHIFTLOCK-C	c
100	64	d		SHIFTLOCK-D	d
101	65	e		SHIFTLOCK-E	e
102	66	f		SHIFTLOCK-F	f
103	67	g		SHIFTLOCK-G	g
104	68	h		SHIFTLOCK-H	h
1D5	69	i		SHIFTLOCK-I	i
106	6A	j		SHIFTLOCK-J	j
107	6B	k		SHIFTLOCK-K	k
108	6C	l		SHIFTLOCK-L	l
109	6D	m		SHIFTLOCK-M	m
110	6E	n		SHIFTLOCK-N	n
111	6F	o		SHIFTLOCK-O	o
112	70	p		SHIFTLOCK-P	p
113	71	q		SHIFTLOCK-Q	q
114	72	r		SHIFTLOCK-R	r
115	73	s		SHIFTLOCK-S	s
116	74	t		SHIFTLOCK-T	t
117	75	u		SHIFTLOCK-U	u
118	76	v		SHIFTLOCK-V	v
119	77	w		SHIFTLOCK-W	w
120	78	x		SHIFTLOCK-X	x
121	79	y		SHIFTLOCK-Y	y
122	7A	z		SHIFTLOCK-Z	z
123	7B				
124	7C				
125	7D				
126	7E				
127	7F		DEL		
128	8D		NUL		
129	81		SOH		
130	82		STX		
131	83		ETX		
132	84		ET		
133	85		ENQ		
134	86		ACK		
135	87		BEL		
136	88		BS		
137	89		HT		
138	8A		LF		

CODE		ASCII		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
139	8B	VT			
140	8C	FF			
141	8D	CR			
142	8E	SO			
143	BF	SI			
144	90	DLE			
145	91	DC1			
146	92	DC2			
147	93	DC3			
148	94	DC4			
149	95	NAK			
150	96	SYN			
151	97	ETB			
152	98	CAN			
153	99	EM			
154	9A	SUB			
155	9B	ESC			
156	9C	FS			
157	9D	GS			
158	9E	RS			
159	9F	US			
160	A0	SPACE			
161	A1	!			
162	A2	"			
163	A3	#			
164	A4	$			
165	A5	%			
166	A6	&			
167	A7	'			
168	A8	(			
169	A9	)			
170	AA	*			
171	AB	+			
172	AC	,			
173	AD	-			
174	AE	.			
175	AF	/			
176	B0	0			
177	B1	1			
178	B2	2			
179	B3	3			
180	B4	4			
181	B5	5			
182	B6	6			
1B3	B7	7			
184	B8	B			
1B5	B9	9			
186	BA	:			
1B7	BB	;			
1BB	BC	<			
1B9	BD	=			
190	BE	>			
191	BF	?			

CODE		ASCII		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
192	C0	@			
193	C1	A			
194	C2	B			
195	C3	C			
196	C4	D			
197	C5	E			
198	C6	F			
199	C7	G			
200	C8	H			
201	C9	I			
202	CA	J			
203	CB	K			
204	CC	L			
205	CD	M			
206	CE	N			
207	CF	O			
208	D0	P			
209	D1	Q			
210	D2	R			
211	D3	S			
212	D4	T			
213	D5	U			
214	D6	V			
215	D7	W			
216	D8	X			
217	D9	Y			
218	DA	Z			
219	DB	[			
220	DC				
221	DD	]			
222	DE	^			
223	DF				
224	E0	`			
225	E1	a			
226	E2	b			
227	E3	c			
228	E4	d			
229	E5	e			
230	E6	f			
231	E7	g			
232	E8	h			
233	E9	i			
234	EA	j			
235	EB	k			
236	EC	l			
237	ED	m			
238	EE	n			
239	EF	o			
240	F0	p			
241	F1	q			
242	F2	r			
243	F3	s			
244	F4	t			

CODE		ASCII		KEY	CHARACTER
Decimal	Hexadecimal	Character	Definition	PRESS	PRODUCED
245	F5	u			
246	F6	v			
247	F7	w			
248	F8	x			
249	F9	y			
250	FA	z			
251	FB				
252	FC				
253	FD				
254	FE				
255	FF				

APPENDIX 3

S P E C I F I C A T I O N S

(a) Computer Unit:

- 6502 and Z-80A CPU's
- 64K RAM
- 26K Bytes of EPROM includes an enhanced Monitor and EC-BASIC
- 40 character X 24 line display (expandable to 80 character X 24 lines); characters formed in a 5 by 7 dot matrix
- Normal, Inverse and Blinking characters.
- Adjustable speaker level
- Full Cursor control and protected screen
- High Resolution Graphics (280 X 192 pixels)
- Seven Expansion Slots
- Intelligent Keyboard with numeric keypad, protected reset, auto repeat function on all keys and upper and lower case shiftlock. Function keys and user-defined keys, battery backed.
- Joystick / Paddle connector
- Plated hole, silk screen, solder mask printed circuit board

(b) Disk Drives:

- Rotational Speed:		300 RPM
- Recording Density (inner track)		5162 bpi
- Flux Density (inner track)		5162 fci
- Track Density		48 tpi
- Media		5 1/4in. diskette
- Capacity		
unformatted		
	(bytes per disk)	218.8K
	(bytes per track)	6.26K
formatted (soft sectoring - 16 sector)		
	(bytes per disk)	143.4K
	(bytes per track)	4096
	(bytes per sector)	256
- Transfer Rate (bits/sec)		250K
- Latency (average)		100 ms
- Access Time		
track to track		20 ms
average		275 ms
setting time		10 ms
- Disk Motor Start Time		1 sec
- DC Voltage requirements		
+12VDC + 5%		0.6 Amp (typical use)
+5VDC +5%		0.2 Amp (typical use)